USER MANUAL

#1, Shivbagh
Ameerpet
Hyderabad- 500016

tel: 040-23737124/25
email: dileepkp@cdac.in

www.cdachyd.in
www.cdac.in

LinK+ Avatar Version: User Manual

Revision 4.0

Authors

Dileep K. Panjala, Suman Rao M, A Raghavendra Rao & Devesh Kumar Gupta
{dileepkp,sumanm,arrao,deveshg}@cdac.in
Team Embedded
C-DAC Hyderabad

Compiled & Modified by

S. Venkata Srikanth & Santosh Sam Koshy
{venkatas,santoshk}@cdac.in
Team Embedded
C-DAC Hyderabad

November 2014

This project is sponsored by Department of Electronics & Information Technology (DeitY), Ministry of Communications & Information Technology, Government of India under the OPA no 32563

Preface

This document contains technical information related to LinK+ IDE which is necessary for the user to

1. Install & use the tool
2. Install additional utilities to use all the features of LinK+ IDE
3. Create & build projects using the tool

User Manual Revisions

Revision	Date	Description
1.0	04/10/2013	Initial Draft of Avatar Release
2.0	17/10/2013	Added a new section for examples & short keys for build options
3.0	05/11/2013	LinK+ First Release: Avatar Version
4.0	25/11/2014	LinK+ Avatar Version to be uploaded in SourceForge

This document is revised periodically; comments and suggestions on all aspects of this manual will be encouraged and appreciated.

Inquiries concerning clarification or interpretation of this manual should be directed to:

C-DAC #1, Shivbagh Satyam Theatre Road Ameerpet, Hyderabad - 500016 Telangana (India) Phones:+91-40-23737124/25 Fax:+91-040-23743382	C-DAC Plot No. 6 & 7, Hardware Park, Sy No. 1/1, Srisailam Highway, Pahadi Shareef Via Keshavagiri (Post) Hyderabad - 500005 Telangana (India)

Contents

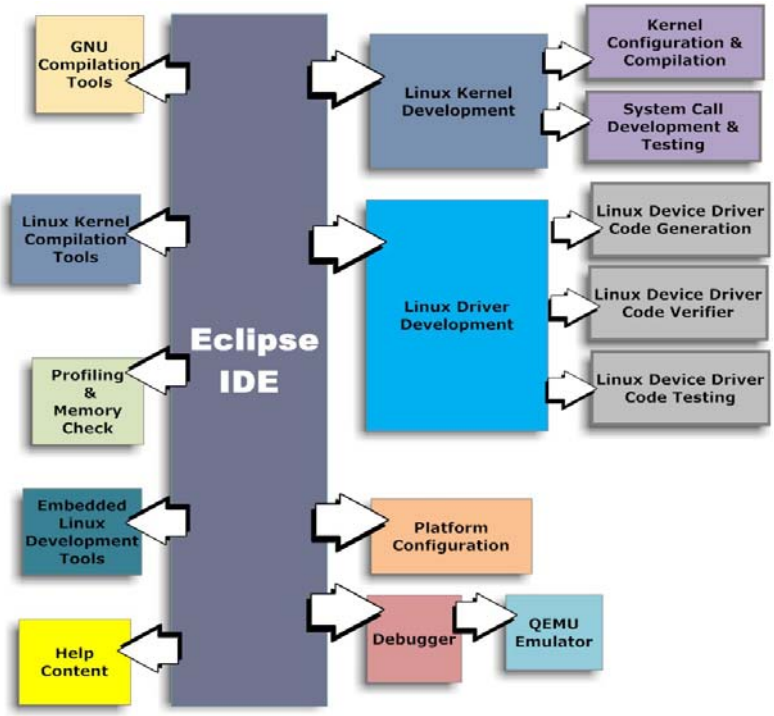
1 About LinK+ IDE 1
1.1 Features 2
2 LinK+ Installer/Plugin Installation 4
2.1 Before You Begin 4
2.1.1 For Ubuntu OS 4
2.1.2 For Other OS 5
2.1.2.1 QEMU installation in CentOS: 5
2.2 Installation of LinK+ IDE 5
2.2.1 LinK+ IDE Installer Installation 6
2.2.2 LinK+ Perspective 8
3 LinK+ Projects 9
3.1 Linux Kernel Configuration and Compilation Project 9
3.1.1 Project Creation 9
3.1.2 Project Build Options 11
3.1.3 Linux Kernel System Call Creation Project 14
3.2 Linux Device Driver Development Project 16
3.2.1 Basic Linux Kernel module program project creation 16
3.2.2 Advanced Kernel Module Program & Device Drivers Project Creation 18
3.2.3 Creating Basic Driver Module 23
3.2.4 Linux Kernel Module Debugging using Probing mechanism 24
3.2.5 Linux Kernel Module program with proc, sys entry creation 25
3.2.6 Project Build Options 33
4 Linux Kernel Compilation Project Preferences 41
4.1 ccache Preference Settings 41
4.2 distcc Preference Settings 43
5 LinK+ Views 48
5.1 Linux Device Driver Example Programs 48
5.2 QEMU Remote Systems 50
6 Linux Kernel Emulation using QEMU 58
6.1 Linux Kernel Emulation using QEMU 58
6.1.1 Linux Kernel Emulation using QEMU with File System 58
6.1.2 Linux Kernel Emulation using QEMU with initrd image 59
6.2 Linux Kernel Debugging using QEMU 60
6.3 System Call Testing with QEMU 62

1

1 About LinK+ IDE

Linux developers are known to be terminal editing geeks, with an inherent liking towards the Linux command line shell and various editors like Emacs, Vi and joe. However, in recent years, major enhancements in the Linux Operating System and its extensive support for various hardware platforms have attracted many more developers, who seek feature-rich, integrated, and user-friendly development environments. To fulfil this need, the Eclipse-IDE provides an excellent platform with extensions such as CDT, Autotools plugin, multiple plugins from the Linux tools project, Target Communication Framework, and much more. The Plug-in Development Environment (PDE) of Eclipse provides tools to create, develop, test, debug, build and deploy plug-ins, fragments, features, update sites and RCP products.

Linux Kernel Programming Plugin Suite (LinK+) is an Eclipse IDE based plugin suite for Linux Kernel Programming that supports X86 & ARM platforms. It provides a feature rich environment that enhances the programming experience of Linux Kernel Developers. The architecture comprises of multiple plug-ins which are built for Eclipse, mainly using “CDT core plugin features”.



1.1 Features

1. **Supports Hardware Architectures**
    a. x86
    b. x86_64
    c. ARM (No support in this release)
2. **Wizard for Kernel Configuration & Compilation**
3. **Supports Multiple Kernel Configuration Options**
    a. xconfig
    b. menuconfig
    c. defconfig
    d. allmodconfig
4. **Supports Multiple Kernel Compilation Options**
    a. bzImage
    b. modules
    c. all
    d. clean
    e. distclean
5. **Reduction in Kernel Compilation Time through**
    a. ccache
    b. distcc
6. **QEMU Emulator support for Linux Kernel Image Emulation**
7. **Linux System Call Development**
    a. Support for adding new System Call with multiple arguments
    b. Emulation of the new System Call with QEMU emulator
8. **Linux Kernel Debugging using QEMU emulator**
9. **Linux Device Driver Auto Code Generation**
    a. UI for Device Driver Auto Code Generation
    b. Support Basic Kernel Module Creations
    c. Support Linux Kernel Features
        i. module_param
        ii. synchronization mechanisms
            1. mutex
            2. semaphores
            3. completions
            4. reader/writer locks
            5. spinlocks
        iii. bottom half techniques
            1. kernel timers
            2. tasklets
            3. workqueues

4. shared queues

iv. Linked List

d. Supports the /proc and /sys file system entry
e. Supports kprobe and jprobe mechanisms
f. Supports IRQ registration and IRQ probing
g. Supports IOCTL commands Creation
h. Supports Character Device Driver type
    i. Supports file_operations selection for Character Device Drivers
    ii. Supports Frameworks
        1. low level
        2. misc registration
        3. tty Framework
        4. UART Framework
        5. Input Framework
    iii. Supports Auto Node Creation
    iv. Supports Ports Registration
i. Supports Block Device Driver (No support in this release)
    i. Supports block_device_operations
    ii. Supports CHS(Cylinder/Heads/Sectors)
    iii. Supports I/O request and make request methods
    iv. Supports Block Framework
j. Supports Network Device Driver (No support in this release)
k. Supports Bus types USB,PCI, I2C, SPI and platform

2

2 LinK+ Installer/Plugin Installation

2.1 Before You Begin

2.1.1 For Ubuntu OS

S.No	Name & Description	Version	Steps to Install
1	**Java Runtime Environment** – Required to execute Java Programs. Is not shipped with Eclipse	>= 1.6.0	*sudo apt-get install openjdk-6-jdk* or Install through **synaptic package manager**
2	**Openssh** – Enables Distributed kernel compilation for Link+ IDE		*sudo apt-get install openssh-server openssh-client* or Install through **synaptic package manager**
3	**Linux Kernel Source and Headers** - Linux Source headers are required for kernel compilation and driver development	Kernel specific	*sudo apt-get install linux-headers-`uname –r`*
4	**Libqt4 Package** – Required by xconfig during kernel configuration		*sudo apt-get install libqt4-dev* or Install through **synaptic package manager**
5	**Libncurses package** – Required by menuconfig during kernel configuration		*sudo apt-get install libncurses5* or Install through **synaptic package manager**
6	**QEMU** – Provides a Linux Kernel Emulation Framework for testing and debugging the Linux Kernel		*sudo apt-get install qemu (for Ubuntu 10.04)* *sudo apt-get install qemu qemu-system (for Ubuntu 12.04)* or Install through **synaptic package manager**
7	**Bridge-utils, IPTables, DNSMasg** – Used by QEMU to provide Network Support		*sudo apt-get install bridge-utils iptables dnsmasq* or Install through **synaptic package manager**

2.1.2 For Other OS

Package	Fedora-19 *$ sudo yum install ...*	BOSS-5 *$ sudo apt-get install ...*	CentOS-6.5 *$ sudo yum install ...*
JRE	java-1.7.0-openjdk-devel.i686	sudo apt-get install openjdk-6-jdk	java-1.7.0-openjdk-devel.i686
OpenSSH	openssh-server openssh-clients	openssh-server openssh-client	openssh-server openssh-clients
Libqt4	qt-devel	libqt4-dev	qt4-devel
Libncurses	ncurses-devel	libncurses5-dev	ncurses-devel
QEMU	qemu-system-arm qemu-system-x86	qemu-system-arm qemu-system-x86	***Refer Section 2.1.2.1***
Bridge-utils, iptables, dnsmasq	bridge-utils iptables dnsmasq	bridge-utils iptables dnsmasq	bridge-utils iptables dnsmasq
g++	Not required	Not required	gcc-c++

** The packages highlighted in blue are already installed by default*

2.1.2.1 QEMU installation in CentOS:

1. Install dependencies :
   $ sudo yum install glib2-devel zlib-devel SDL-devel libtool libtool-ltdl-devel

2. Download QEMU-1.6.0
   $ wget -c http://wiki.qemu-project.org/download/qemu-1.6.0.tar.bz2

3. Extract and get inside qemu-1.6.0
   $ tar xvf qemu-1.6.0.tar.bz2
   $ cd qemu-1.6.0

4. Configure QEMU
   $ ./configure --target-list=arm-softmmu,i386-softmmu,x86_64-softmmu –enable-sdl

5. Build QEMU
   $ make -j4

6. Install QEMU
   $ sudo make install

2.2 Installation of LinK+ IDE

Installation of LinK+ IDE can be done by using the Automated Installer *(bundles Eclipse IDE & LinK+ Plugins into a single application)*

2.2.1 LinK+ IDE Installer Installation

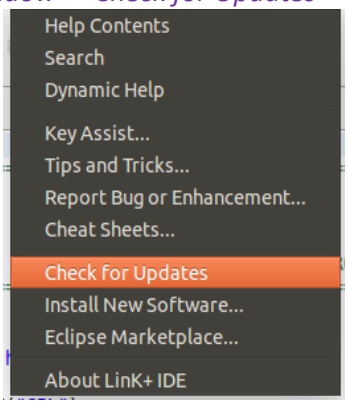
1. **Download** the **LinK+ IDE Installer**.
   The Installer is in **.tar.gz** compressed format.

2. To **uncompress** the LinK+ IDE Installer
   tar –xzvf <LinK+ Installer File Name>

This will create a folder named **'LinK+ IDE'**. Open the folder. You will find an executable titled '**linkplus**'. Run the executable, which will open eclipse with support for LinK+ IDE plugins enabled.

2.2.2 LinK+ IDE Updates :

To update LinK+ IDE click on *Window -> Check for Updates*



2.2.3 Installing LinK+ Plugins

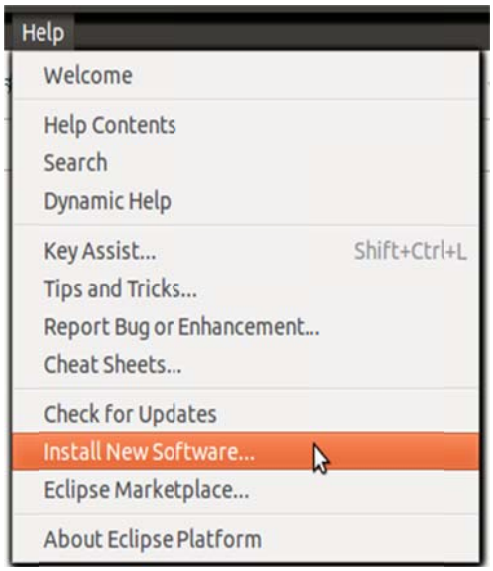
Before Installing the LinK+ plugin, you should have a recent version of Eclipse cpp of any one of the eclipse variants **helios (>= 3.6 ver)** already installed on your PC.

Eclipse can be downloaded from **www.eclipse.org/downloads/**

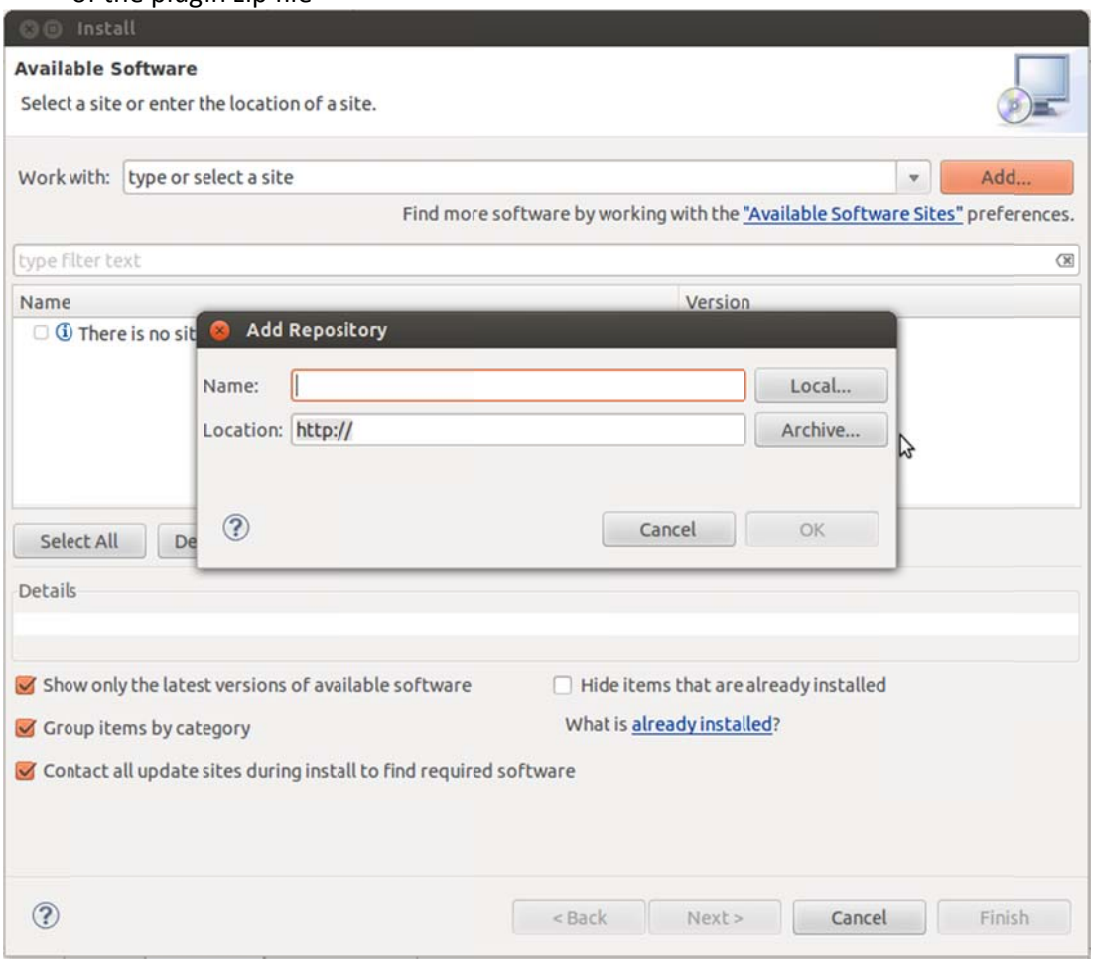
> **Note:** *In this release (Avatar), plugins are developed only for Eclipse Helios variant*

After Eclipse IDE is installed, LinK+ IDE Plugins can be integrated in the following manner

1. **After Starting Eclipse:** Click on *Help->Install New Software*



2. Enter the location of the plugin zip file by clicking on **Archive** and selecting the **Path** of the plugin zip file



Note: *Follow the instructions of the install wizard. The warning about the installation of an unsigned feature can be ignored. Eclipse has to be restarted after installation*

Using LinK+ Update Site: Enter the LinK+ Update site URL

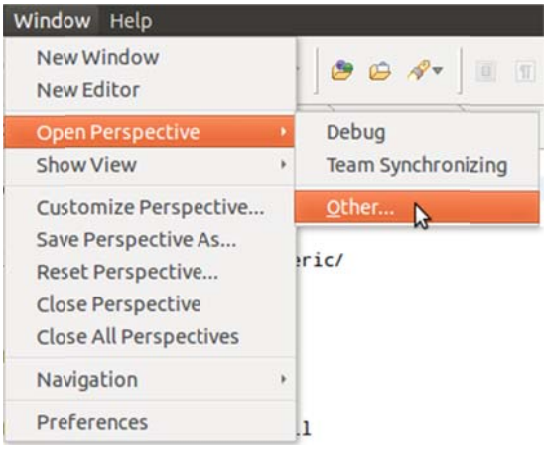
https://sourceforge.net/projects/linkplustest/files/repository/

2.2.4 LinK+ Perspective

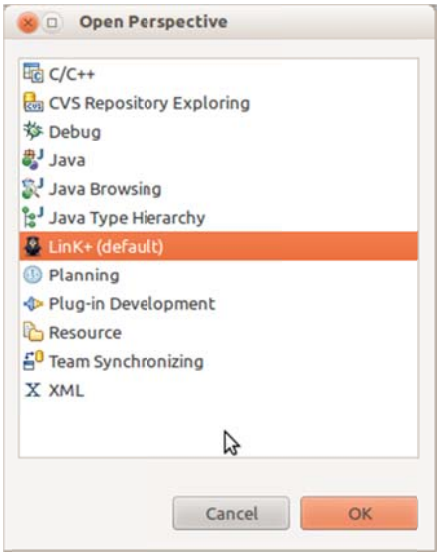
LinK+ Perspective is a customized perspective of Linux Kernel Source Configuration & Compilation and Device Driver Development.

When the user is in a different perspective, they can directly goto the LinK+ Perspective by

Window->Open Perspective->Other…



This will open up a wizard, select **LinK+ (default)** to activate the Link+ perspective



The perspective is now changed from C/C++ Perspective to LinK+ (default) Perspective.

Note: *LinK+ Perspective is an extension of C/C++ Perspective*

3

3 LinK+ Projects

3.1 Linux Kernel Configuration and Compilation Project

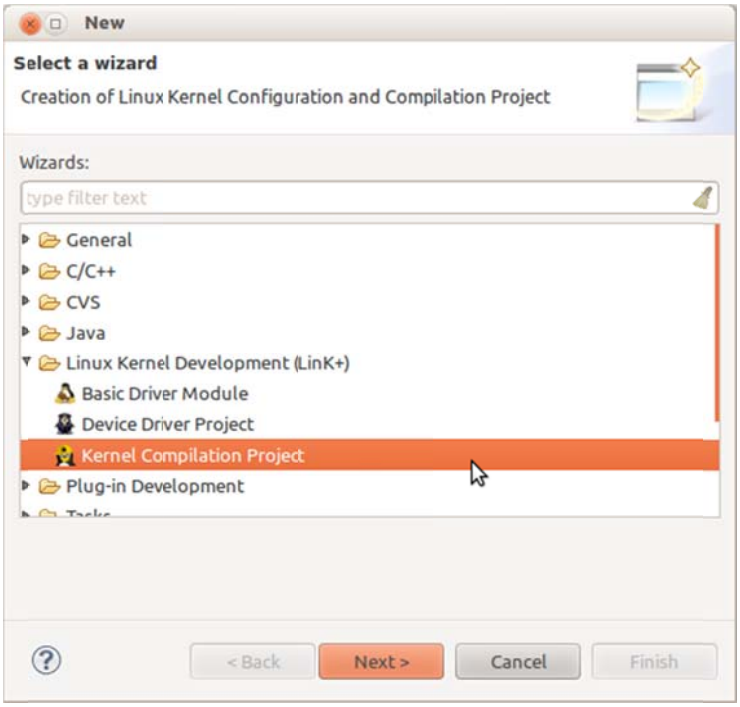
3.1.1 Project Creation

3.1.1.1 Linux Kernel Compilation Project

Linux Kernel Configuration and Compilation Projects are created by selecting

File-> New-> Project…

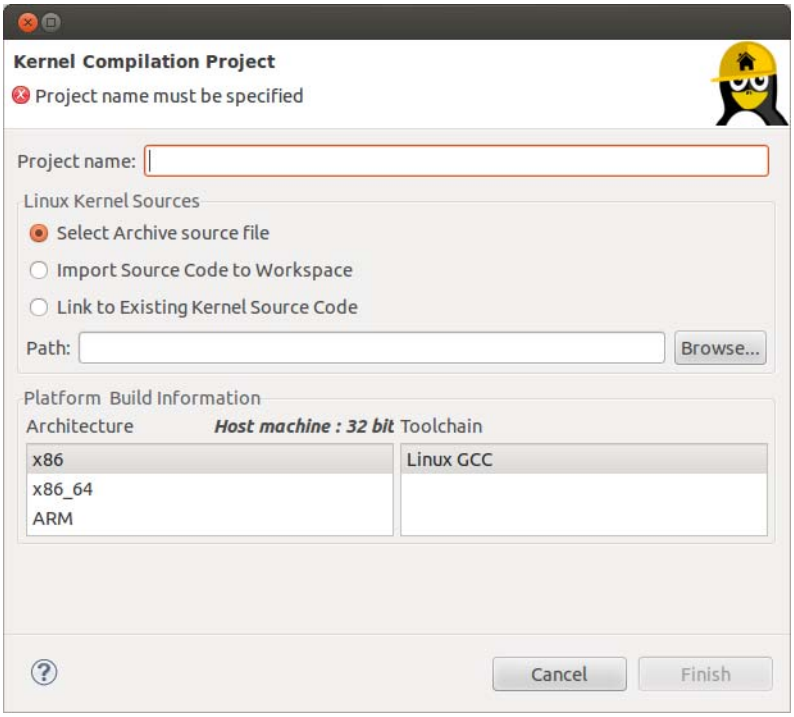
Click on the option, **Linux Kernel Development (Link+),** to expand the option**.**
Click on **Kernel Compilation Project** to select the option



Click on **Next ->** and follow the instructions of Project Creation.

The Kernel Compilation wizard will open where the user can enter any **Project Name**

Then the user needs to select the **Linux Kernel Source.** Multiple options are provided where the user can select one among them based on his requirement.



1. **Select Archived Source Code:** This option is to select an archived kernel source with extension formats supporting .tar.gz/bz2/xz. The project is created in the workspace and the kernel source files are uncompressed into the project

2. **Import Source Code to Workspace:** This option copies already uncompressed kernel source, available on the disk, to the Project Workspace. The user can navigate to the path of the required kernel source, which is thereafter copied to the project workspace.

3. **Link to Existing Kernel Source Code:** This option enables the user to link to an alternate location on the disk, apart from the project workspace, to add kernel source code. This is essentially important if the user does not want to create a new project in the workspace and he wants to operate from the kernel source original location only

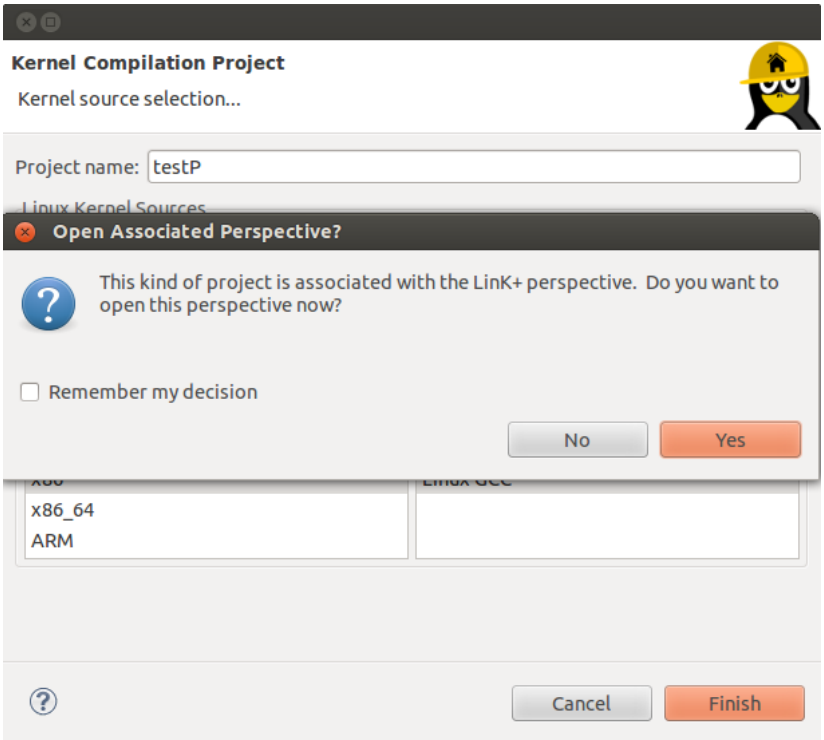
Depending on the selection of Linux Kernel Source, the user can choose the **Path** of the Kernel Source by clicking the **Browse** button

Then the tool will automatically select the hardware architecture based on the host system configuration

Click **Finish** Button to complete the process.

> **Note:** *LinK+ IDE supports multiple architectures (x86, x86_64 and ARM). In this release (Avatar), ARM hardware architecture is not supported.*

After completion, when the user is in a different perspective, they can directly goto the LinK+ Perspective as shown below



Click **Yes** to change to **LinK+ IDE Perspective**

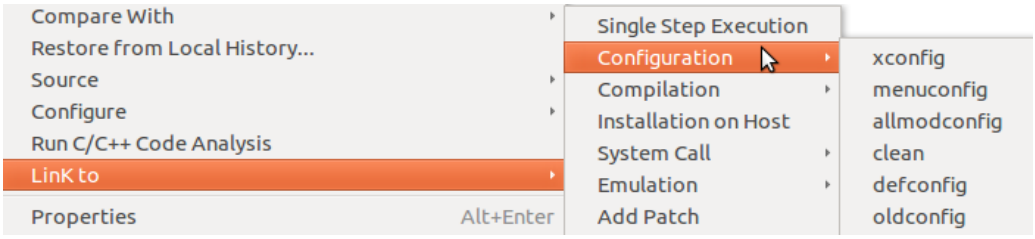
3.1.2 Project Build Options

Project build options are following

- Configuration
- Compilation
- Installation

3.1.2.1 Configuration

Right click on the respective project in the **Project Explorer View** then goto *LinK to ->Configuration*



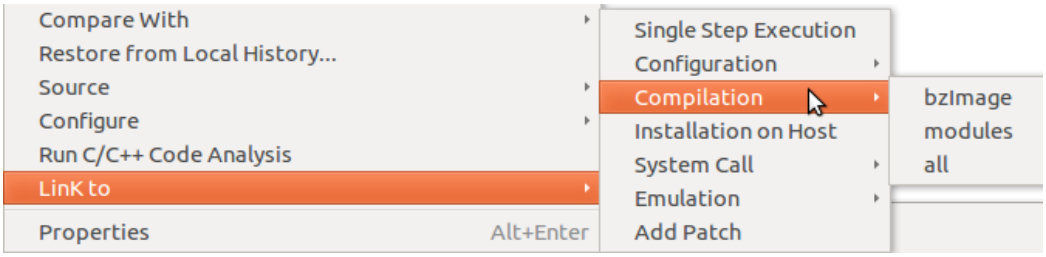
The wizard provides the following options. User can choose any option based on his requirement.

- xconfig
- menuconfig
- allmodconfig
- clean
- defconfig
- oldconfig

3.1.2.2 Compilation

Right click on the respective project in the **Project Explorer View** then goto
LinK to -> Compilation

The wizard provides the following options. User can choose any option based on his requirement.



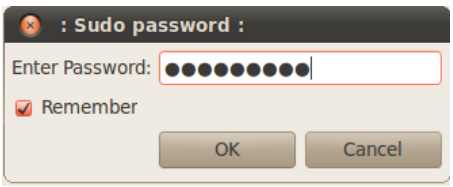
Note: *Generally compilation takes some time based on your host system hardware configuration. In order to reduce time use **distcc** and **ccache** features in preference page. By this feature user can complete the compilation process in less time. For distcc & ccache features (**refer section 4)** of this user manual*

3.1.2.3 Installation on host

Note: *Before Installation on host, the user needs to compile **modules.***
*After completing compilation Right click on the respective project in the **Project Explorer View** then goto LinK to -> Compilation ->modules*

For the **Installation on host**, the user after completing compilation of modules
Right click on the respective project in the **Project Explorer View** then goto
LinK to -> Installation on host

It will open one window for providing password. Enter **Password** to install on host.

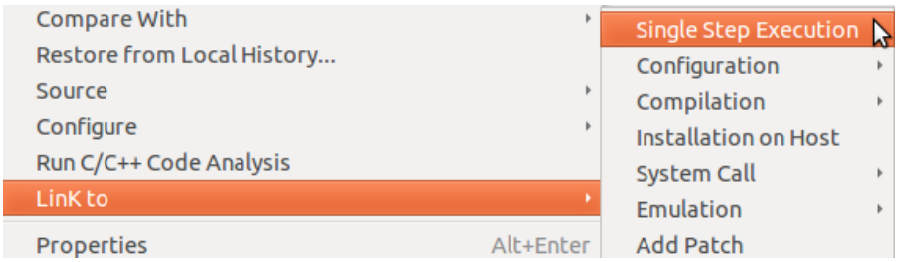


3.1.2.4 Single Step Execution

Linux kernel Configuration & Compilation Build Project can be done in a single step with the following wizard.

Right click on the respective project in the **Project Explorer View** then goto

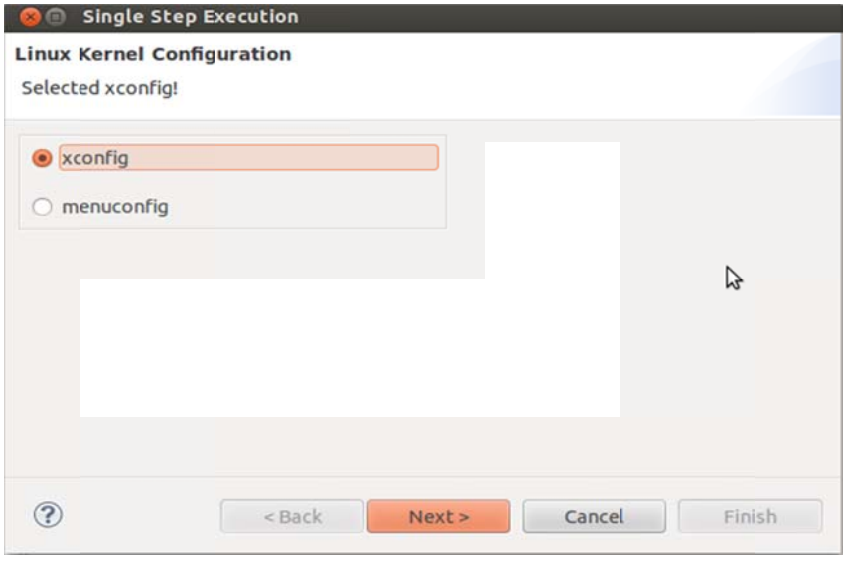
LinK to -> Single Step Execution



It will open a wizard with three pages, first page has **configuration options**, second page has **compilation options** and third page has **Installation on Host** or emulate with **QEMU file system** of your kernel source

3.1.2.4.1 First Page: Linux Kernel Configuration

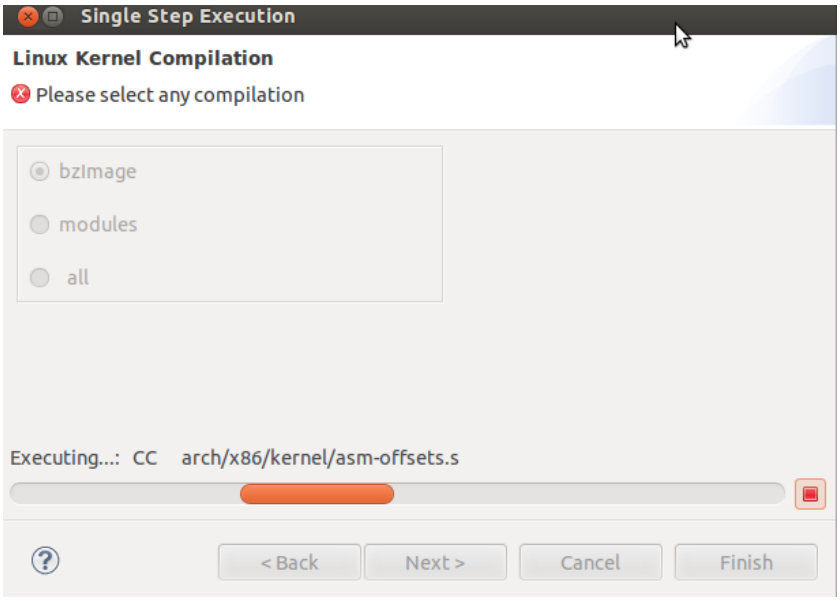
Select **xconfig** or **menuconfig** to perform configuration



Click on **Next** Button to perform configuration operation and enter the second page

3.1.2.4.2 Second Page: Linux Kernel Compilation

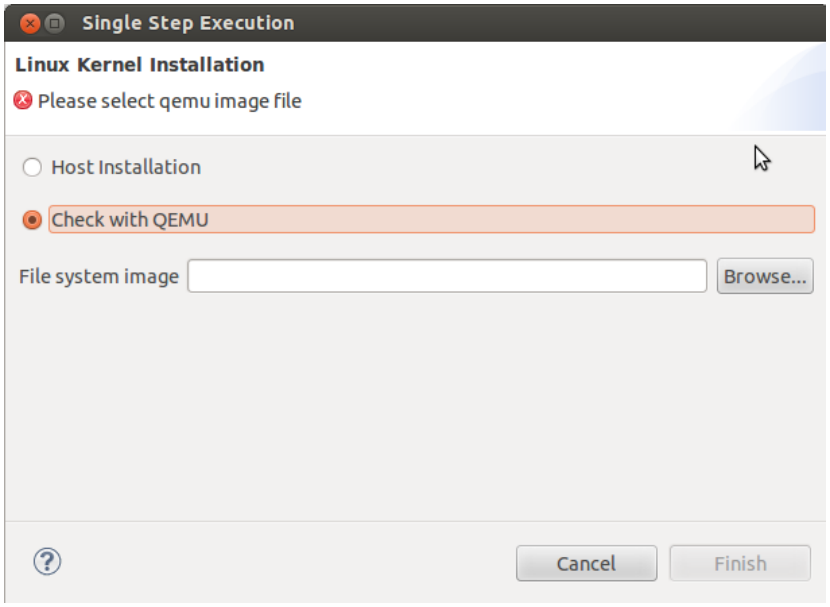
Select any one option given in the wizard below according to requirement



Click on **Next** button to complete this operation and go to the third page

3.1.2.4.3 Third Page: Linux Kernel Installation

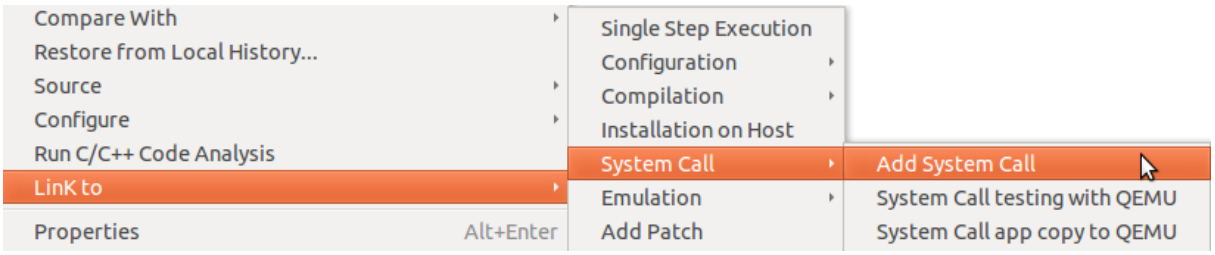
Select installation type**,** if **Host installation** provide **password**
or
If you want to **Check with QEMU**, then **Browse** for QEMU file system **Path** & then click on **Finish** button to perform operation



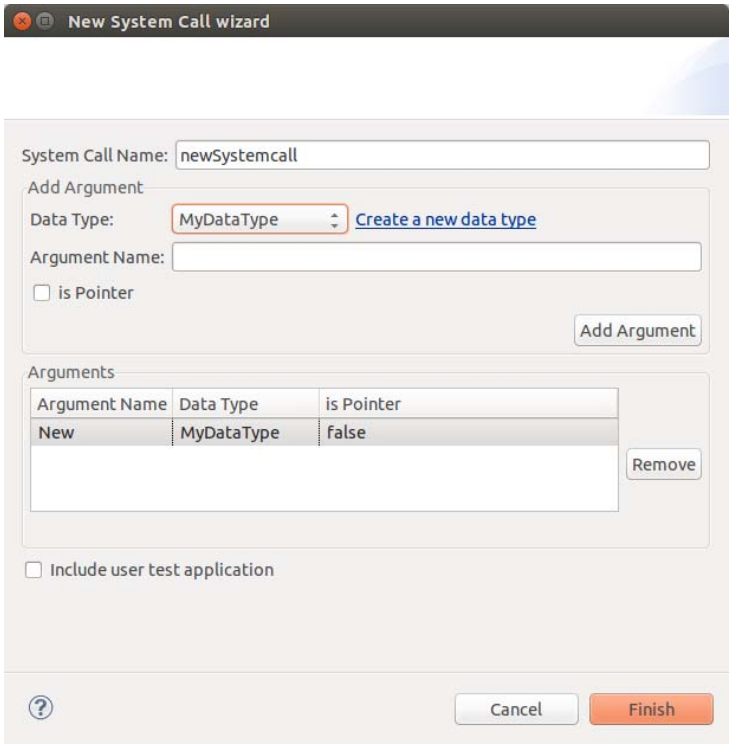
3.1.3 Linux Kernel System Call Creation Project

System Call creation can be done in the following way
Right click on the respective project in the **Project Explorer View** then goto

LinK to -> System Call -> Add System Call

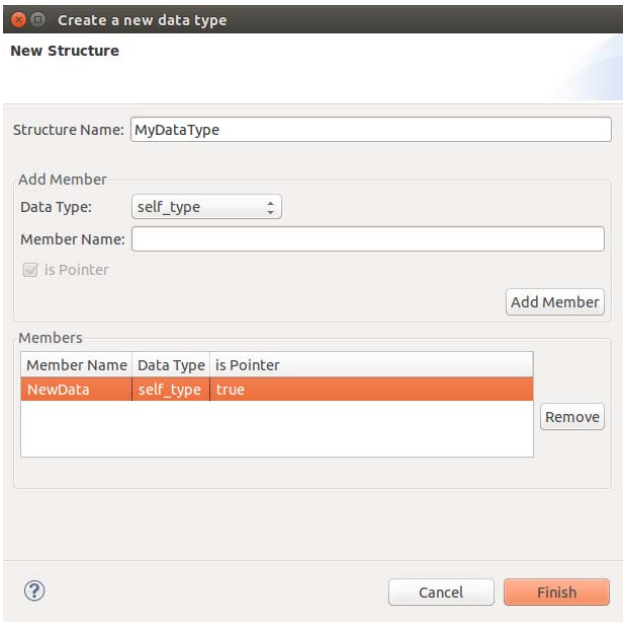


After clicking on **Add System Call** it will open a wizard.



Enter **System call name**, then select existing **Data Type** and finally click the **Add Argument** button

Also, the user can create a new data type by clicking **Create a new data type** link as shown in Figure. If you click on the link it will open the following window.



After that, in **Structure Name** field enter name of your new data type
Then, in order to add members into the above structure goto **Add Member** field, select **Data Type** and give the data type name.
Next, click on **Add Member** button and repeat this process for adding more members.

Finally, click on **Finish** button.

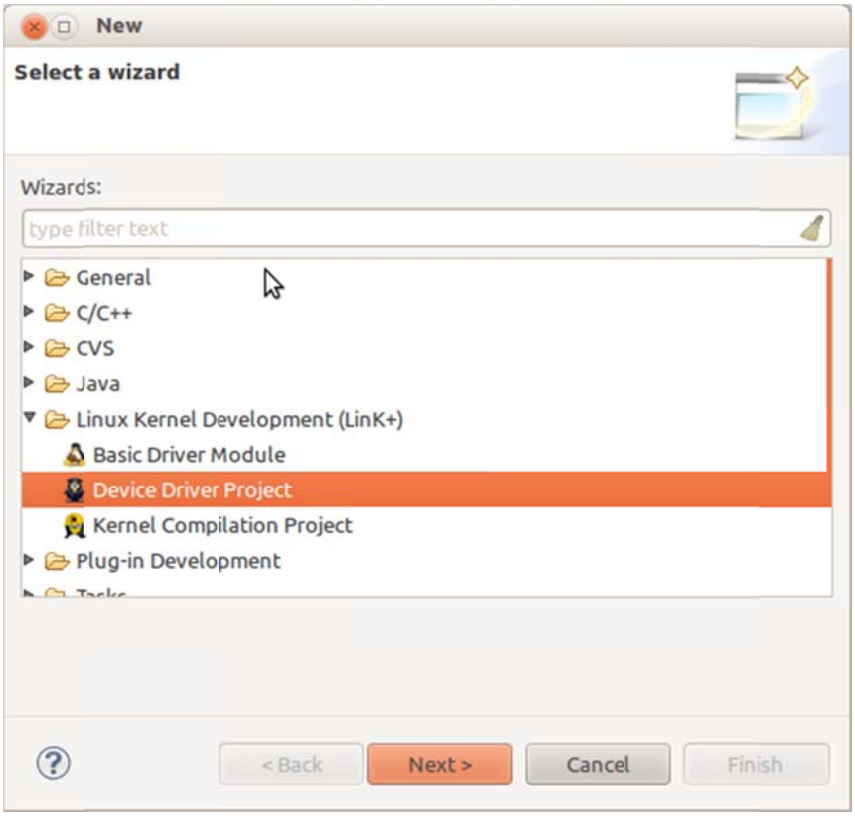
> **Note:** *If you want to test your System Call with System Call application enable Test user application. It will create a user space application for the System Call testing.*

3.2 Linux Device Driver Development Project

3.2.1 Basic Linux Kernel module program project creation

Linux Device Driver Projects are created by selecting
File->New->Project…

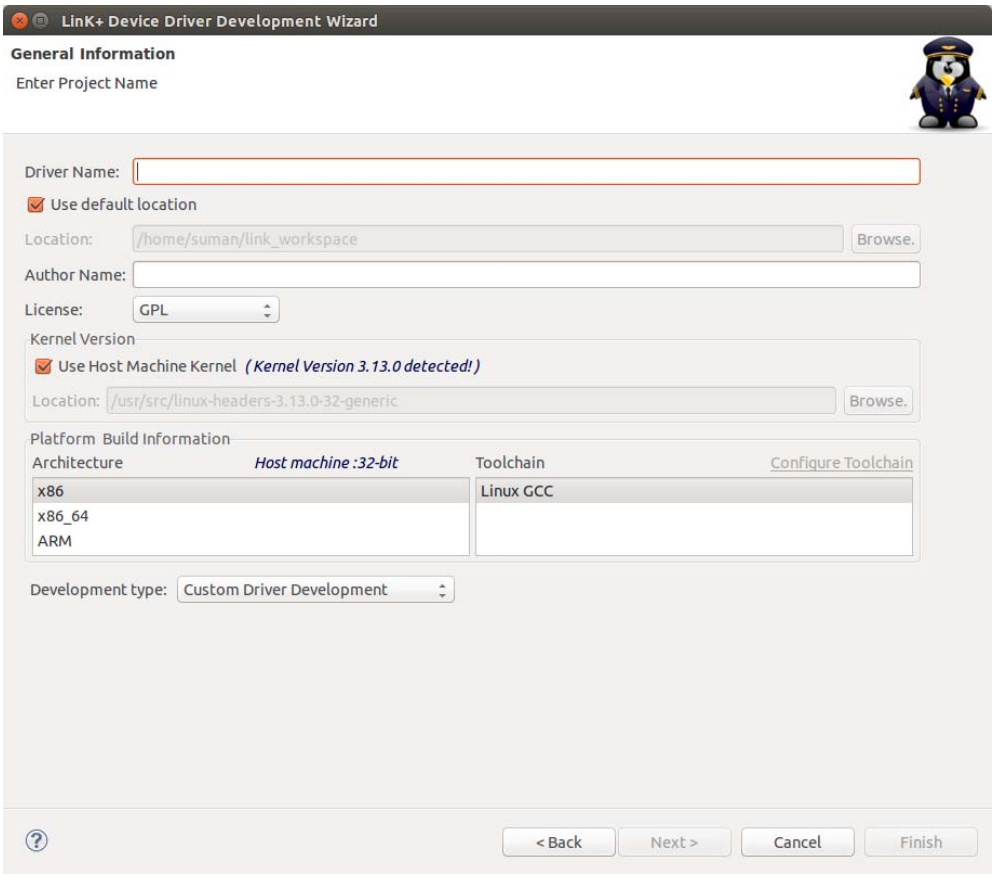
then Click on that it will open New Project wizard, in that go to **Linux Kernel Development (Link+)** & select **Device Driver Project**



Note: *This wizard contains three pages General Information Page, Device Information Page & Driver Information Page. User is free to finish the process at any of the wizard page.*

General Information Page contains the basic information like,

- In the **Driver Name** field, type a name for the driver project, such as *my_driver*.
- When **Use default location** is selected, the New Project Wizard will create a new project with the specified name in the workspace. Otherwise, you can specify the location, Uncheck the default location and Click on **Browse** to browse for a location in the file system.



- In the **Author Name** field, type a name
- Select the **license** type from the License combo box
- When **Use Host Machine Kernel** is selected, the New Project Wizard will use the running kernel source location. Otherwise, you can specify the other linux kernel source location, **Uncheck Use Host Machine Kernel** and **Click on Browse** to browse for a location in the file system.
- Select the architecture from the **Architecture** list.
- Select a required toolchain from the **Toolchain** list.

In the **Development Type** dropdown box**,** the user will have 2 options

1. **Custom Driver Development**
2. **Typical Driver Development**

Click on **Next->** for other Kernel Development Features
or
Click on **Finish** for basic module programming

Note: *In this Release (Avatar), we are not supporting ARM Architecture*

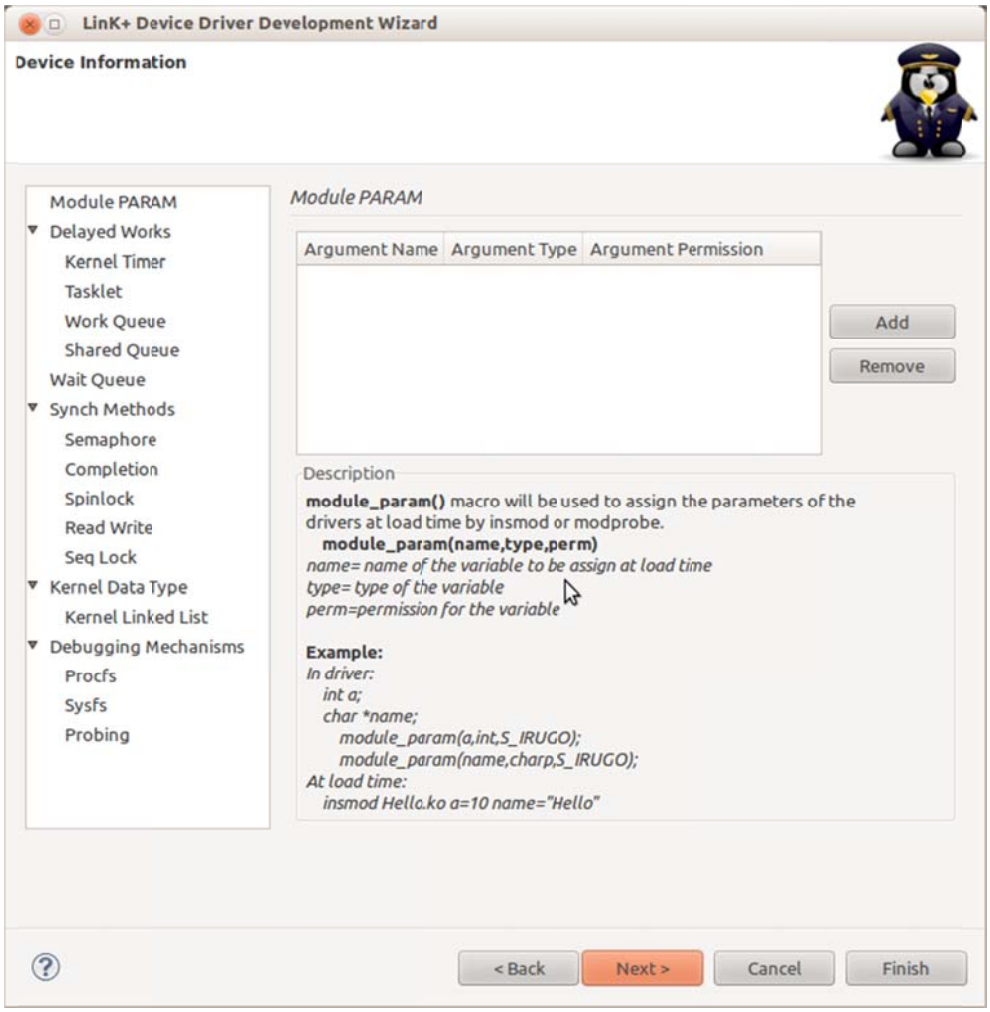
dure below:

3.2.2 Advanced Kernel Module Program & Device Drivers Project Creation

Linux Kernel Features Page contains the Kernel features like

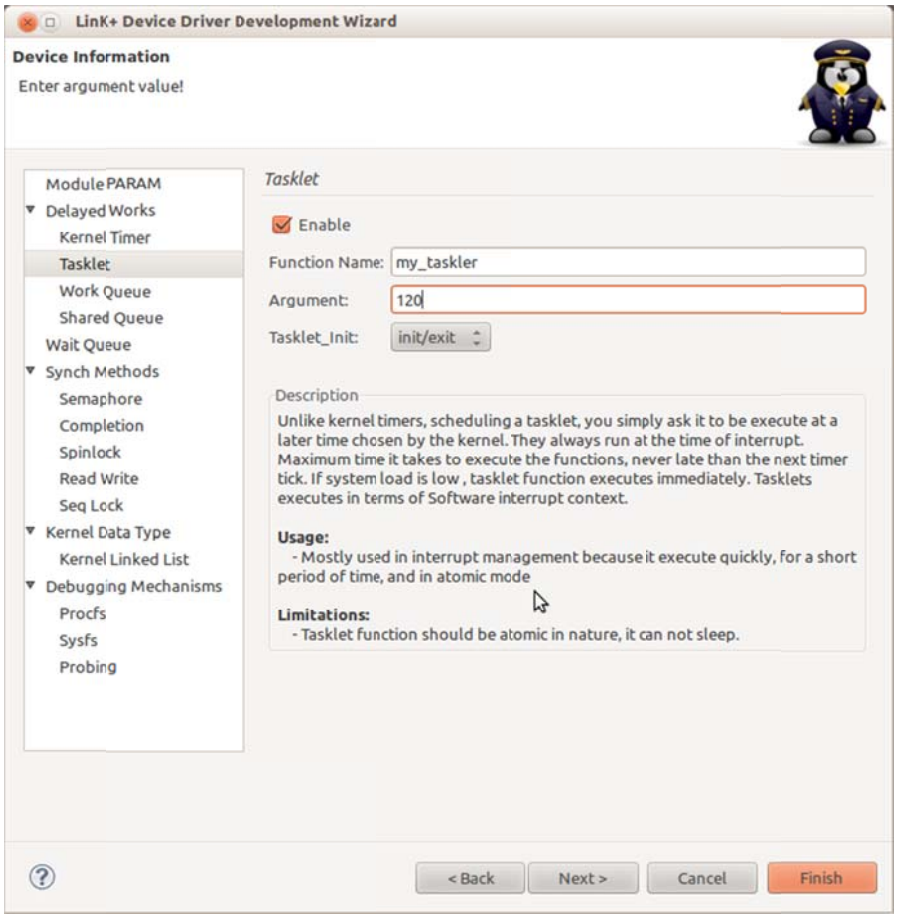
- Module Param
- Delayed Works ->Kernel Timer, Tasklet
- Synchronization->Semaphore, Spinlock and many more!
- Kernel Data Types
- Debugging Mechanisms

All the features are optional & user can select any number of features



Example: Tasklet Feature:

Once the user selects the **Tasklet Feature** on the left side, the tasklet wizard will ask for the necessary fields as shown in the screenshot below:


LinK+ Device Driver Development Wizard
Device Information
Enter argument value!
Module PARAM
Delayed Works
Kernel Timer
Tasklet
Work Queue
Shared Queue
Wait Queue
Synch Methods
Semaphore
Completion
Spinlock
Read Write
Seq Lock
Kernel Data Type
Kernel Linked List
Debugging Mechanisms
Procfs
Sysfs
Probing
Tasklet
Enable
Function Name: my_taskler
Argument: 120
Tasklet_Init: init/exit
Description
Unlike kernel timers, scheduling a tasklet, you simply ask it to be execute at a later time chosen by the kernel. They always run at the time of interrupt. Maximum time it takes to execute the functions, never late than the next timer tick. If system load is low , tasklet function executes immediately. Tasklets executes in terms of Software interrupt context.
Usage:
- Mostly used in interrupt management because it execute quickly, for a short period of time, and in atomic mode
Limitations:
- Tasklet function should be atomic in nature, it can not sleep.
< Back
Next >
Cancel
Finish

Click on **Next->** for the Device Driver Features
or
Click on **Finish** button for the project creation with the above features

Linux Kernel Features Page is divided into two parts:

3.2.2.1 First Part

First half contains Driver Type, Bus Type, Device Type and Framework Method

- In **Driver Type** field select the driver name from the list.

Note: *In this Release (Avatar), only char and block driver are supported*

- In **Bus Type** field select the bus name from the list.

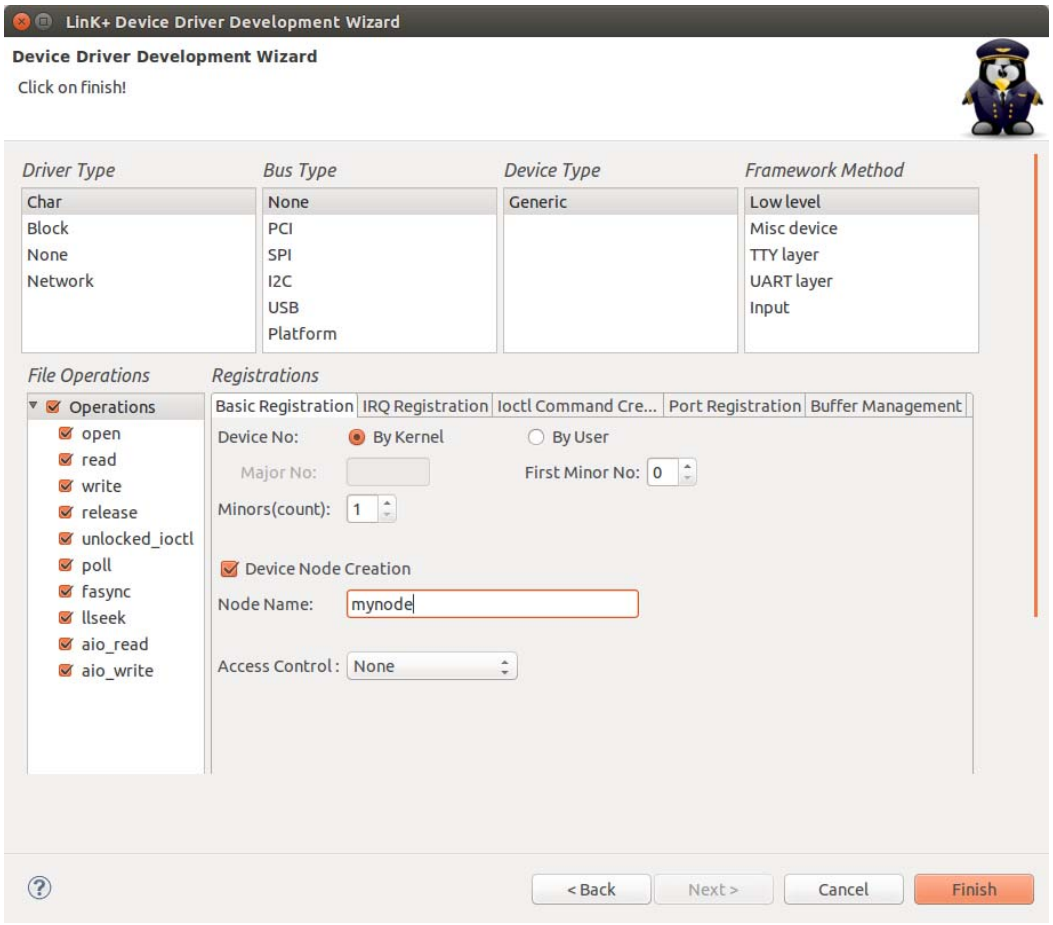
Note: *In this Release (Avatar), only bus type none is supported*

- In **Device Type** field select the device name from the list.

Note: *In this Release (Avatar), only device type generic is supported*

- In **Framework Method** field select the framework method name from the list.

> **Note:** *The frameworks will dynamically change based on the Drive Type selection*



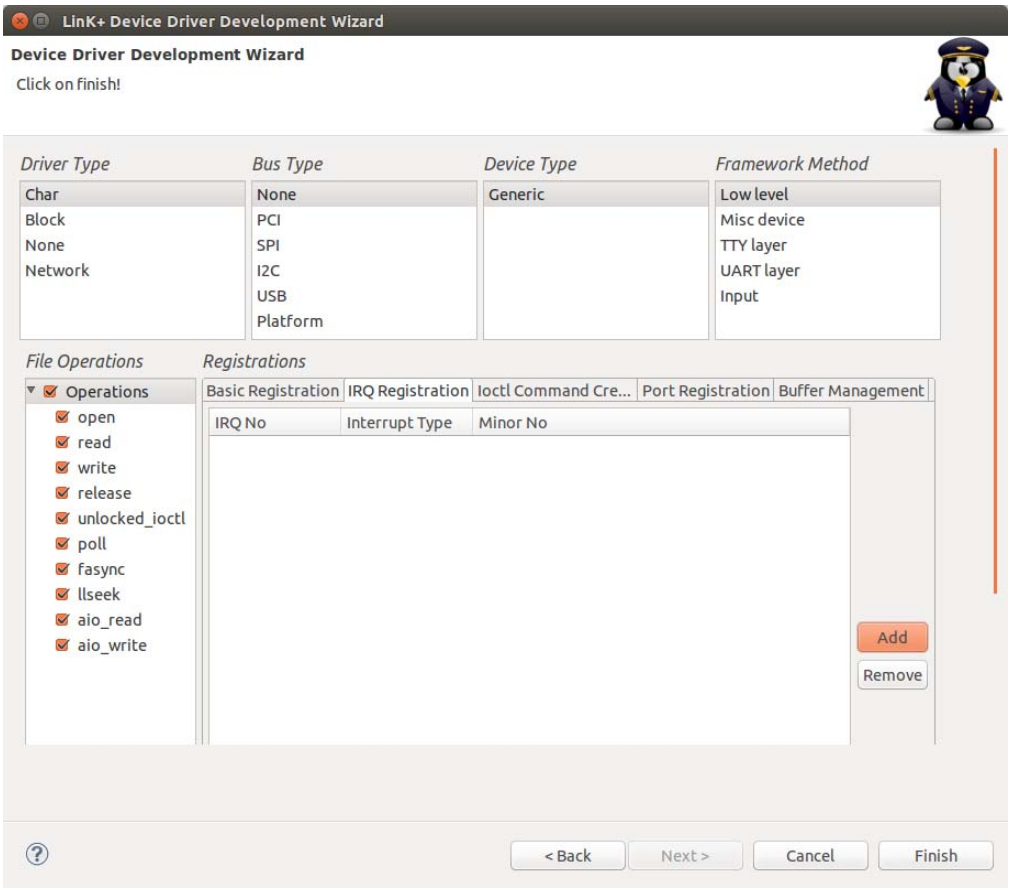
> **Note:** *Second half view is dynamic it depends on the user's first selection.*

3.2.2.2 Second Part

Second half view contains

Operations: It contains a list of functions. You can select zero or more functions from the list.
Registration View: It depends on the user's first half selection.
Basic Registration tab fields are compulsory and other registrations fields are optional.

Example: IRQ Registration View



Click on **Finish** button to generate the Device Driver code.

A device driver project is created with the default settings and a full set of configurations based on the project type and toolchain you selected. Basic Device Driver Code is generated based on your selection
You should now see the new project in C/C++ Projects view.

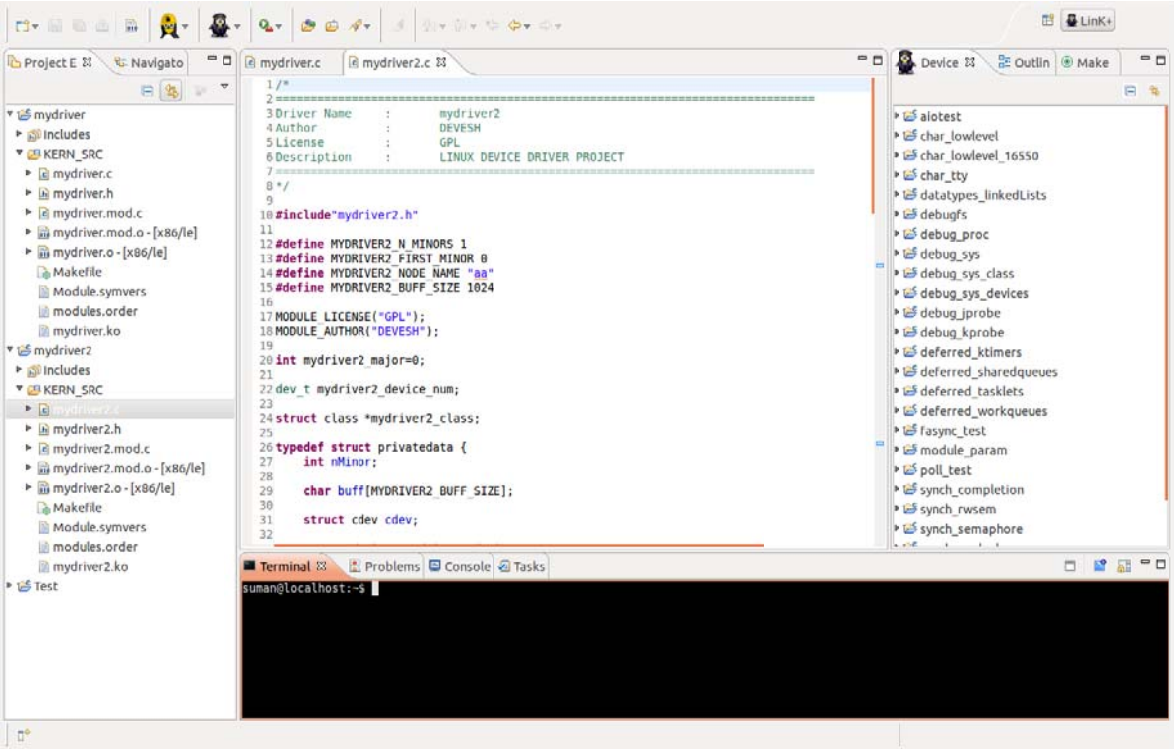
If the user chooses **Typical Driver Development** type, follow the given procedure below:

Unlike custom driver development typical driver development asks only driver configuration, According to the driver configuration it generates code with required kernel features.

- To select an item double click on it.
- Select framework type, driver type and bus type from framework list, driver list and bus list respectively.
- Set no of devices if applicable.

Click on **Finish** button.

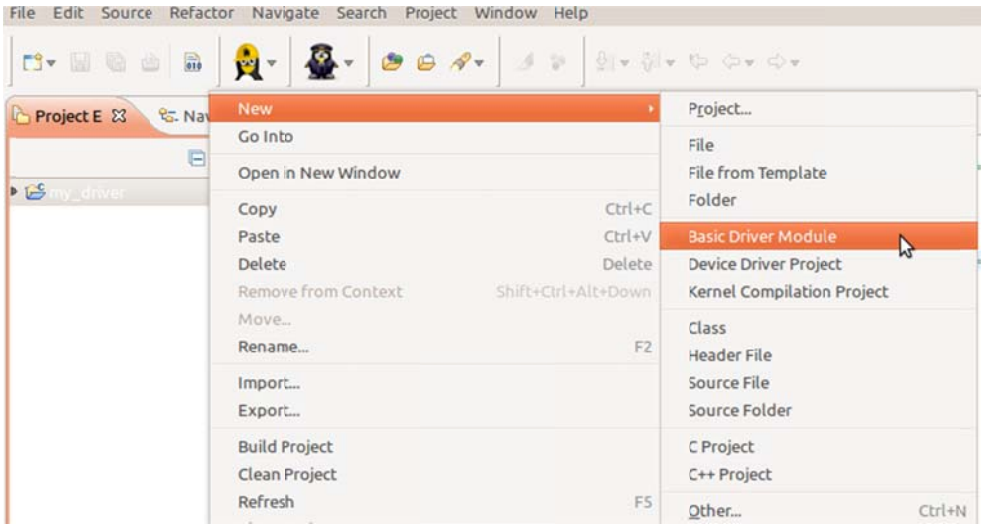
Eclipse View:



3.2.3 Creating Basic Driver Module

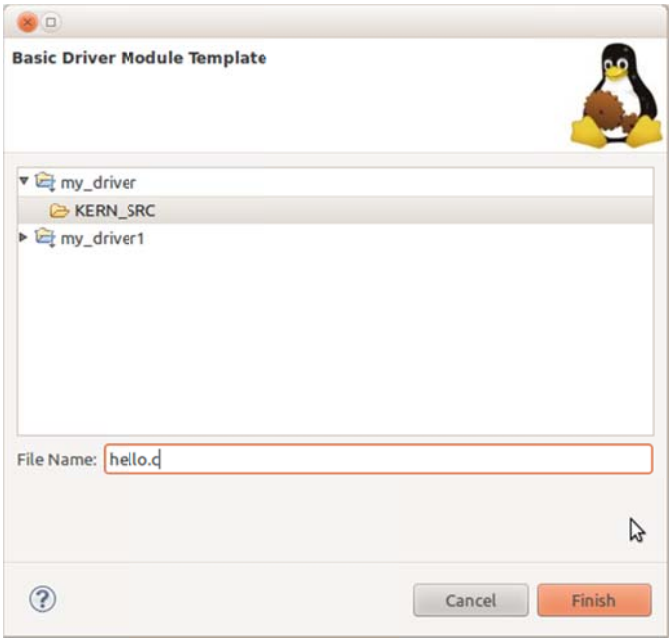
After the user creates a driver module, he can extend the project by creating a basic driver module through; In the project explore, **Right Click** on the respective project,

Goto *New->Basic Driver Module* as shown in the screenshot below



It will open a wizard; the user needs to enter a **File Name** to be created in the KERN_SRC as

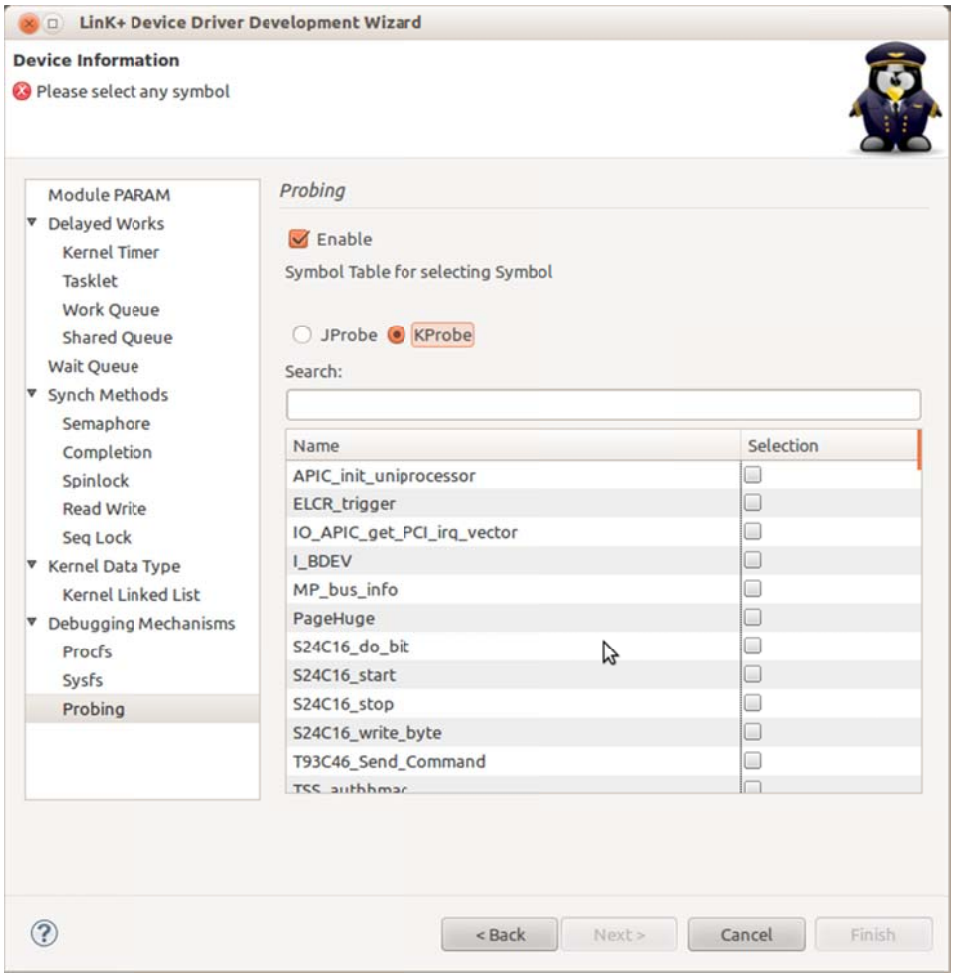
shown in the screenshot below



3.2.4 Linux Kernel Module Debugging using Probing mechanism

The page can be obtained either during the project creation (***refer section 3.2.1***) or while extending the project in order to develop the probe modules. In this tool we are providing both jprobe & kprobe features.

Select *Debugging Mechanisms->Probing* on the left pane and check the **Enable** checkbox on the right pane, as shown in the below screenshot



Then the user can select either **JProbe** or **KProbe** by clicking the radio button as shown in the screenshot below

After selecting the probing mechanism, the user needs to select the required symbol by entering the names in the **Search** textbox

The wizard will display the symbols based on user search. Now the user has to select required symbols by clicking the **Selection**

Click on **Finish** button to complete.

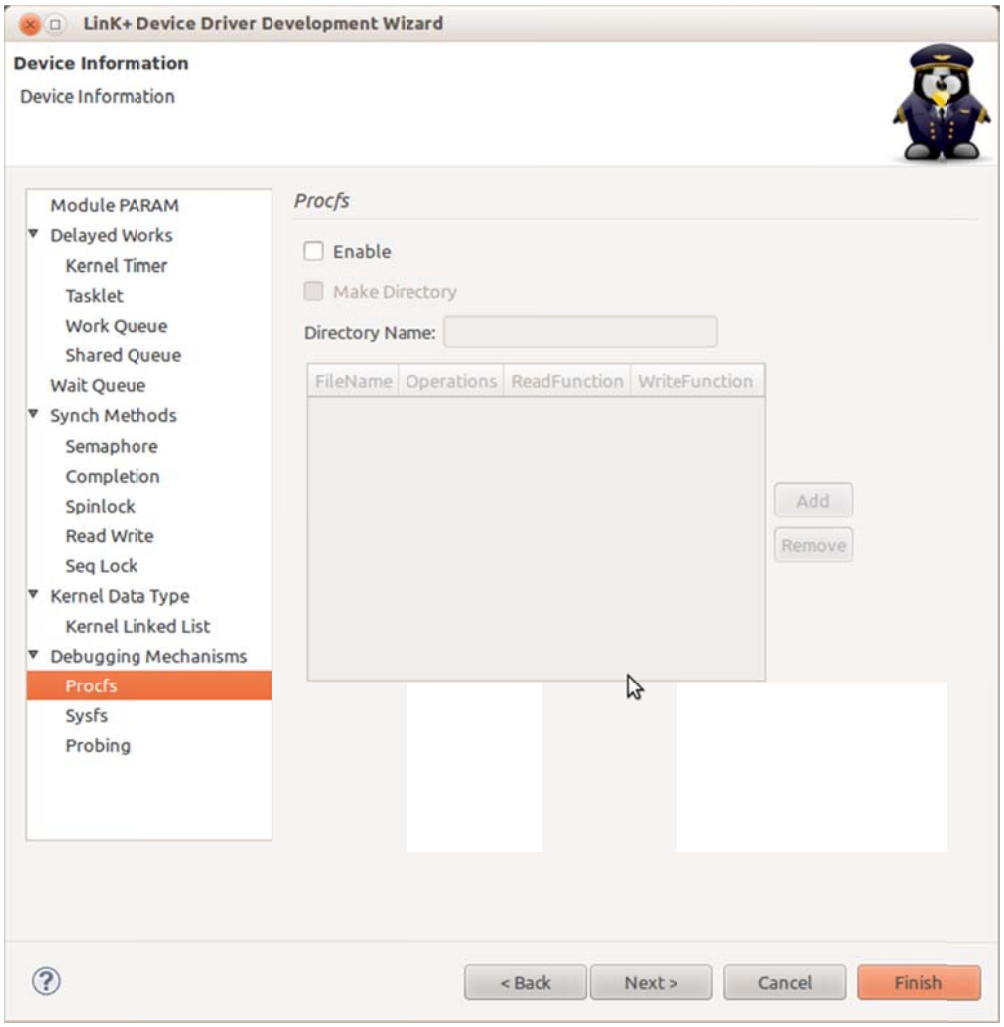
3.2.5 Linux Kernel Module program with proc, sys entry creation

3.2.5.1 /proc

The /proc file system is a virtual file system that permits a novel approach for communication between the Linux Kernel and the user space. In the procfs, virtual files can be read from or written to as a means of communicating with entries in the kernel.

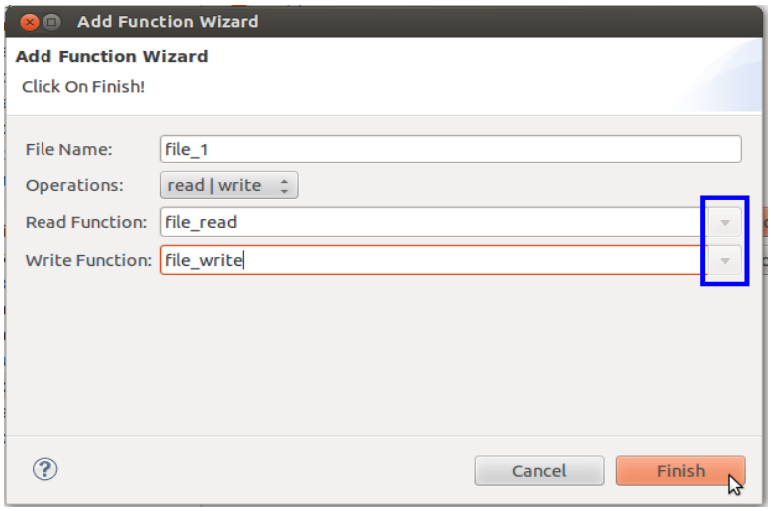
The page can be obtained either during the project creation (*refer section 3.2.1*) or while extending the project. In order to access the procfs,

Select *Debugging Mechanisms->Procfs* on the left pane and check the **Enable** checkbox on the right pane, as shown in the below screenshot



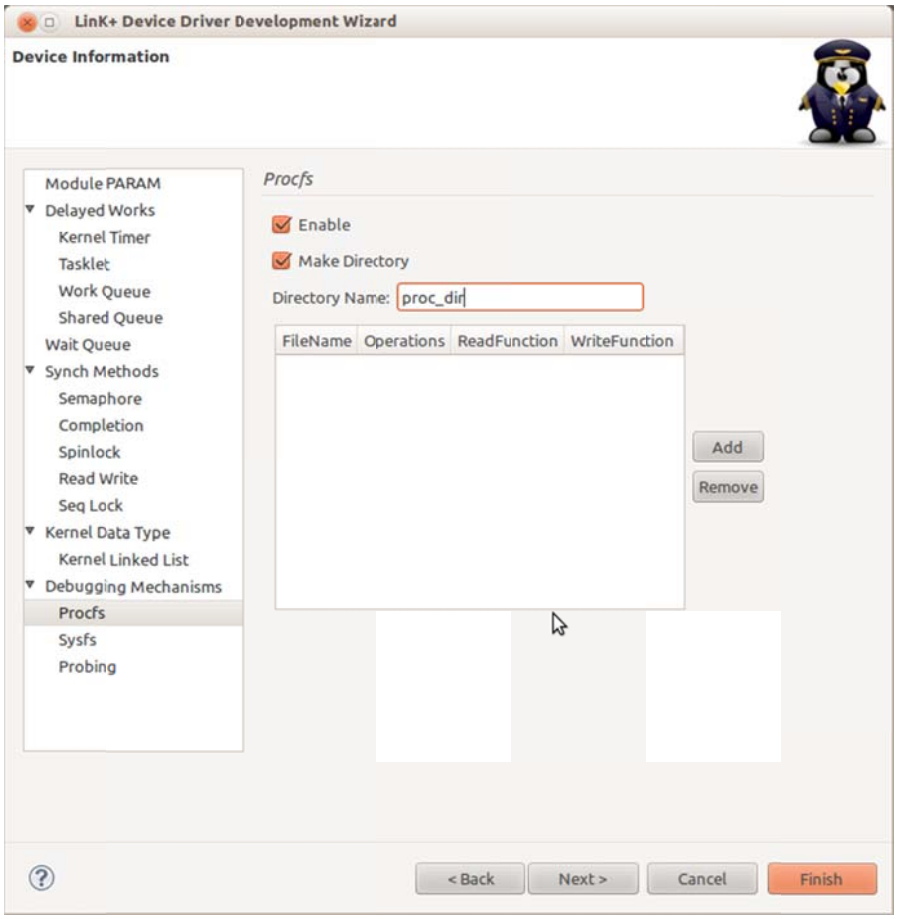
Files can be added to the procfs by mentioning a list of files in the table provided. In order to do so, click on the **Add** button

Upon clicking the **Add** button, the following window will popup:
File Name: Name of the file you wish to create
Operations: Select the operations that you want to perform on this file. Possible options are read and read | write.
Read Function: Give the name of the function to be invoked when read() is called on this file. If you wish to go with any of the previous read functions, select the arrow (highlighted in the figure) and choose one from the list.
Write Function: This field is visible only for *read | write* operations. If enabled, give the name of the function to be invoked when write() is called on this file. If you wish to go with any of the previous write functions, select the arrow (highlighted in the figure) and choose one from the list.



Once everything is done click on the **Finish** button and the file will be added to the list.

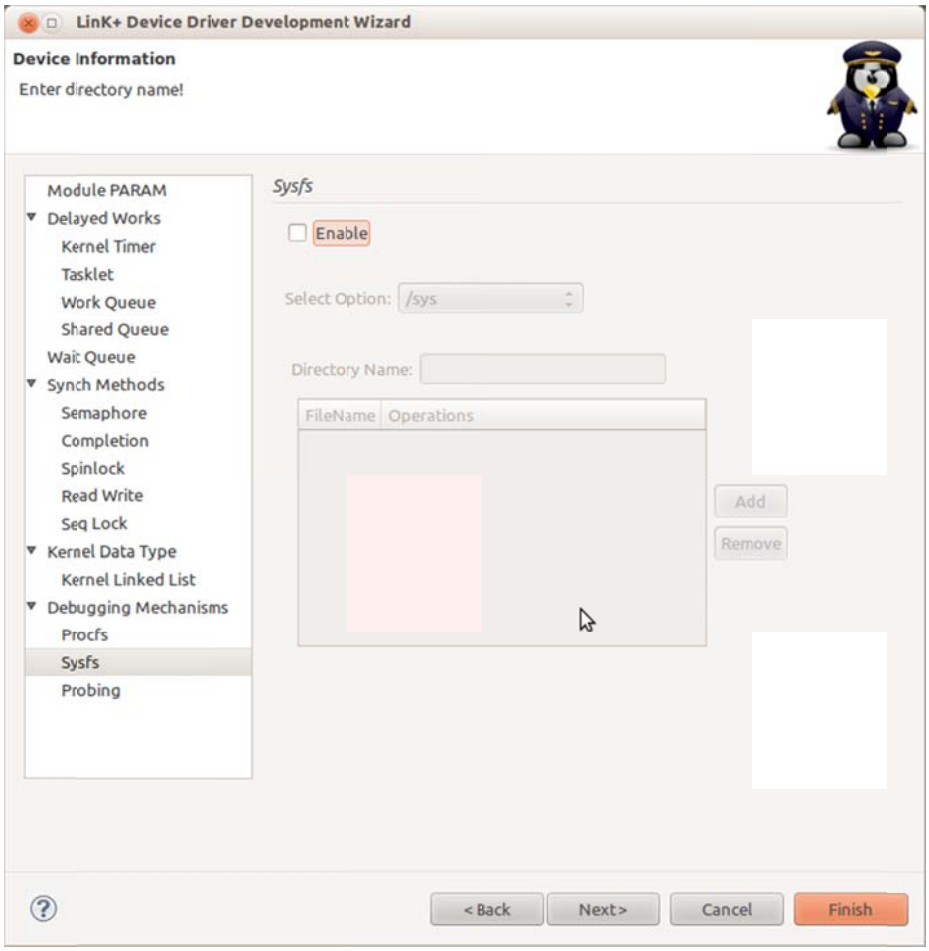
Make Directory checkbox will be used to create a directory in the /proc file system with the name given by the user. Now the above setup generates code to create a directory as /proc/proc_dir and within this directory, the file file_1 is created upon which both read and write operations are possible



3.2.5.2 /sys

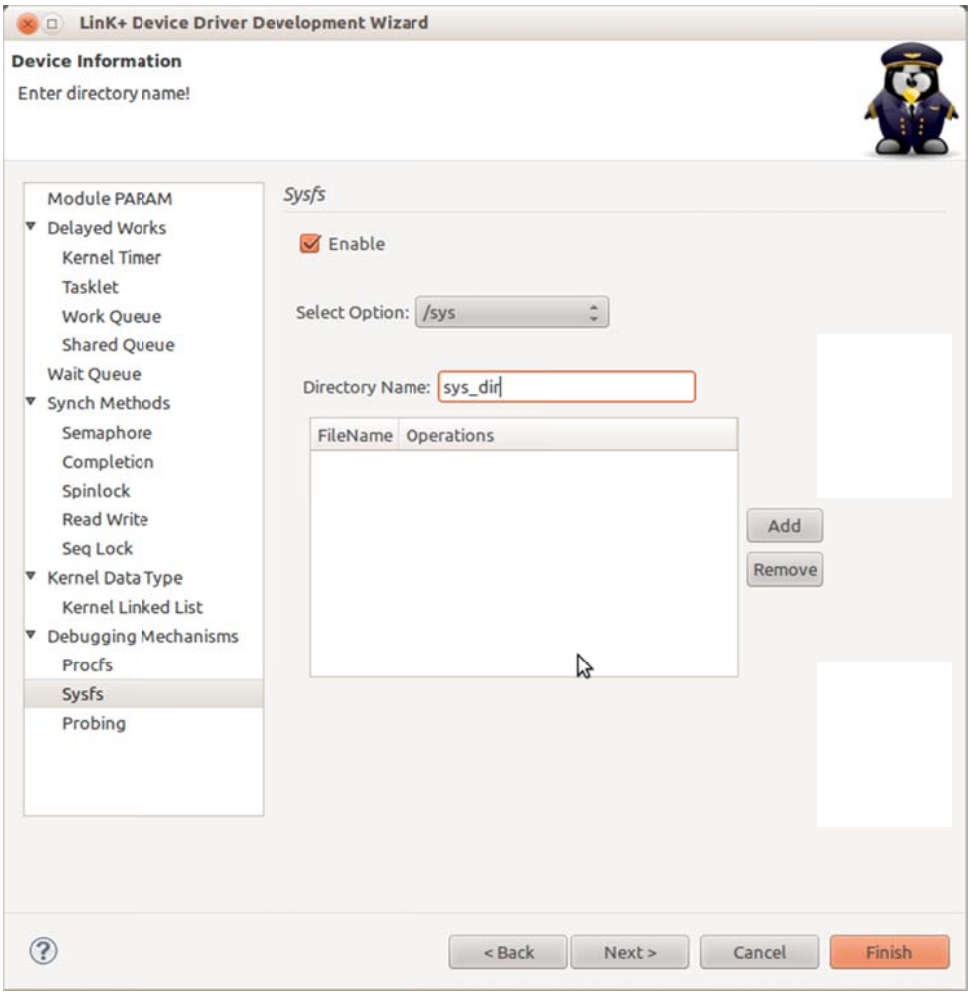
The /sys file system is a virtual file system over which the information about the devices and drivers are scattered throughout the file system. This part of the wizard provides the inputs to the user through which the user can get hands on to accessing the device model of the Linux kernel.
The page can be obtained either during the project creation (***refer section 3.2.1***) or while you are extending the project. In order to access the sysfs

Select *Debugging Mechanisms->Sysfs* on the left pane and check the **Enable** checkbox on the right pane, as shown in the below screenshot.

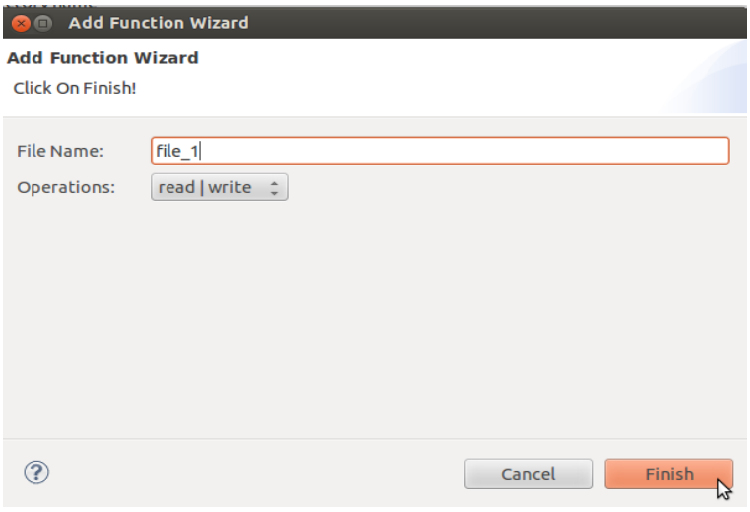


Once you've enabled the sysfs, **Select Option** dropdown will list the sysfs entries. User can select any one of the below options as shown in the screenshot

- /sys
- /sys/class/
- /sys/device
- /sys/ kernel/debug

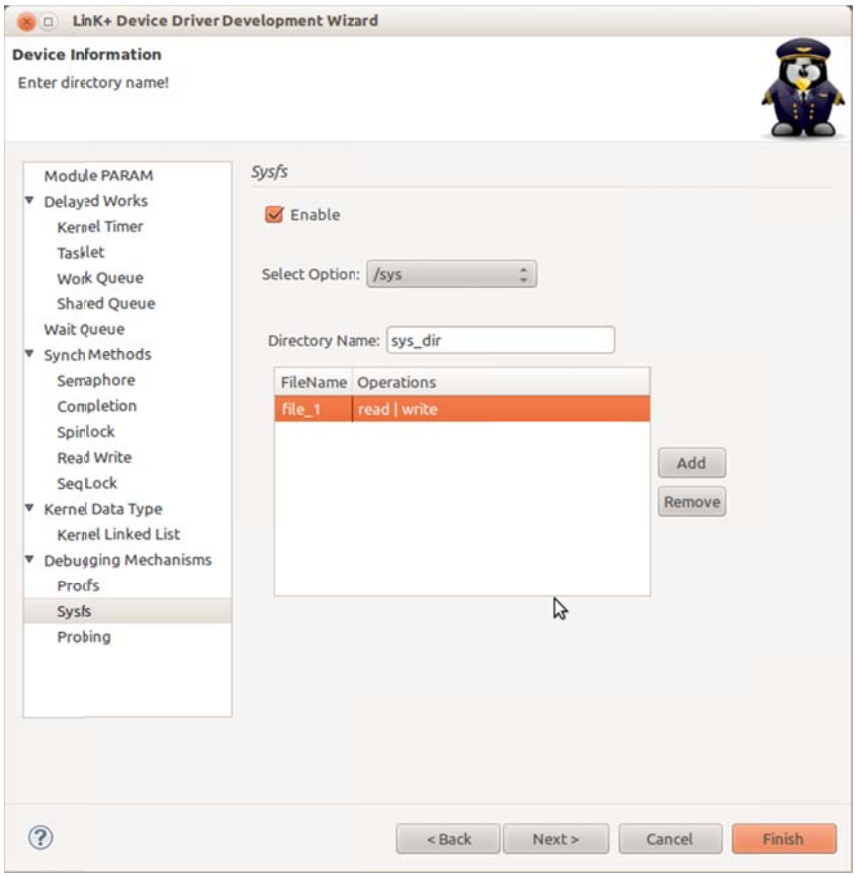


Inorder to add files in the /sys, click the **Add** button which will open the following popup:



File Name: Name of the file to be created
Operations: Mention the operations that are to be permitted on this file. The possible options are read and read | write.

Click on the **Finish** button. The files get added in the table as shown below:



Note: *The above procedure is common for /sys/class & /sys/device*

3.2.5.3 /sys/kernel/debug

The file system is also known as debugfs. debugfs is a simple memory-based file system, designed specifically to debug Linux kernel code . debugfs helps kernel developers export large amounts of debug data into user-space.

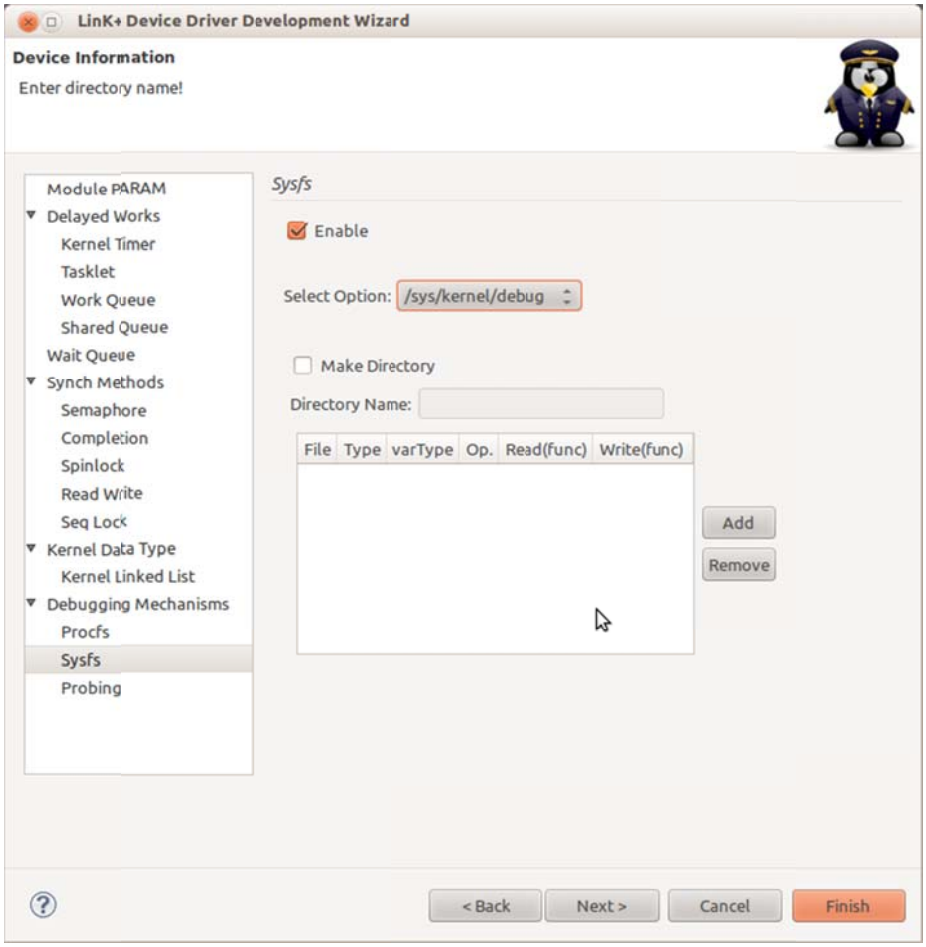
Before proceeding further, check if debugfs is mounted in your system by,

 # mount | grep debugfs
none on /sys/kernel/debug type debugfs (rw)

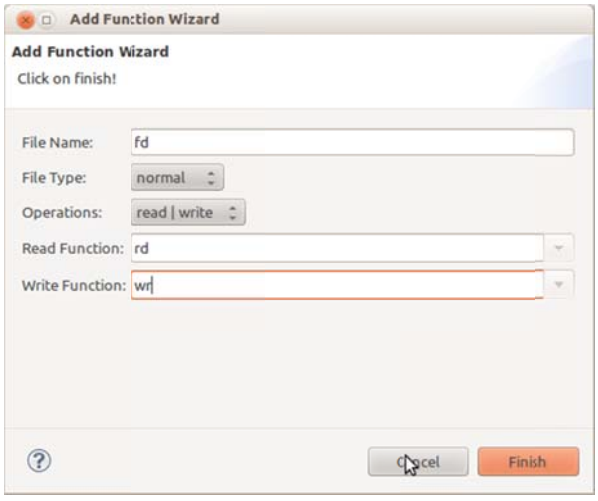
If you see output as above, you have debugfs pre-mounted. If not, you can mount it (as root) with the command shown below:
 # mount -t debugfs nodev /sys/kernel/debug

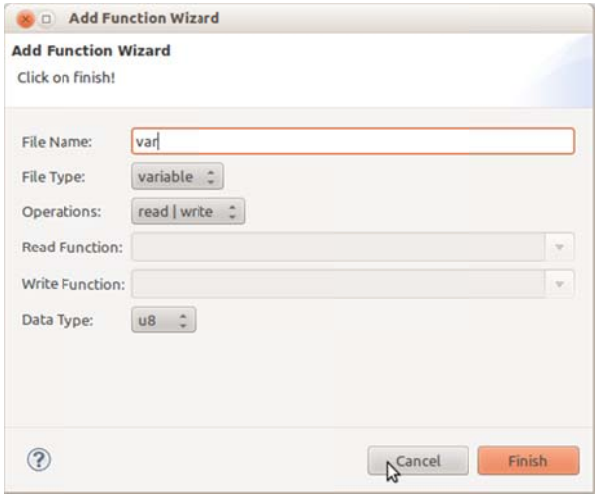
the page can be obtained either during the project creation (***refer section 3.2.1***) or while you are extending the project. In order to access the Sysfs,

Select *Debugging Mechanisms->Sysfs* on the left pane and check the **Enable** checkbox on the right pane, as shown in the below figure. In this case, the option to be selected is */sys/kernel/debug*.



In order to add files in the /sys/kernel/debug, click the **Add** button, which will open the following popup:



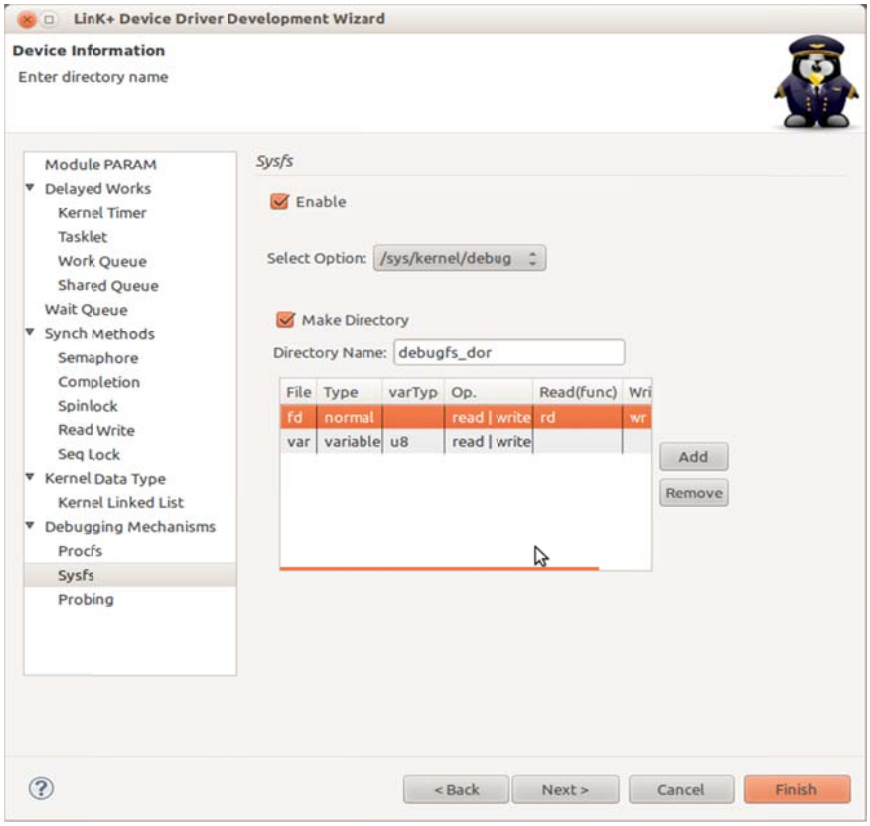


File Name: The name of the file to appear in debugfs
File Type: The type of the file to be created. The possible options are:

- **normal :** a normal file will be created to which read and write functionalities can be applied
- **variable:** a variable file will be created upon which read and write operations will modify the contents of the variable specified. For this option a variable with the same name as that of the file name will be created in the code.

Operations: Mention the operations that are to be permitted on this file. The possible options are read and read | write.
Read Function: Visible when File Type is specified as normal. Specify the name of the read function, to be called when read () is called on this file. If you wish to go with any of the previous read functions, select the arrow (highlighted in the figure) and choose one from the list.
Write Function: Visible when File Type is specified as normal and when read | write operations are permitted. Specify the name of the write function, to be called when write () is called on this file. If you wish to go with any of the previous write functions, select the arrow (highlighted in the figure) and choose one from the list.
Data Type: Visible when File Type is specified as variable. Specify the size of the variable associated with this type.

Once done with filling the details, click on the **Finish** button and the files will be added to the table. Now the above setup generates code to create a directory as ***sys/kernel/debug/debugfs_dir*** and within this directory, the files *fd* and *var* are created upon which both read and write operations are possible.

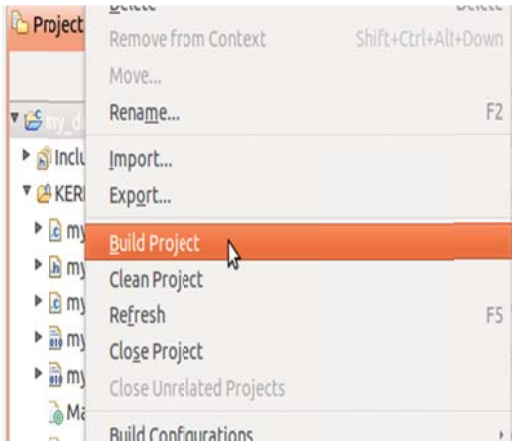


3.2.6 Project Build Options

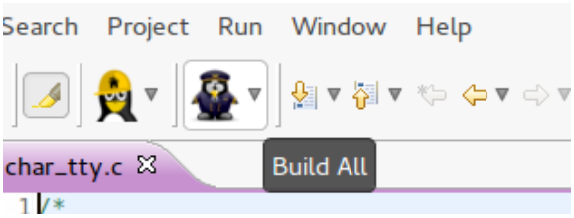
By default, **Build Automatically** is enabled (if you save the project files it will build automatically) and if you want to disable this feature select

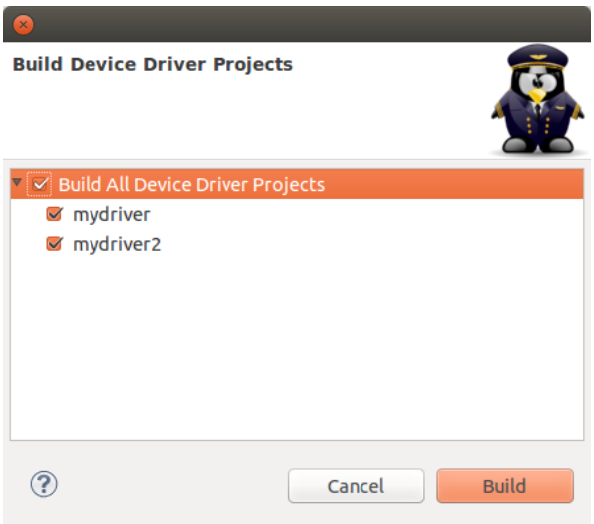
Project -> uncheck the *Build Automatically*

To build a project right click on that project, then click on **Build Project.**

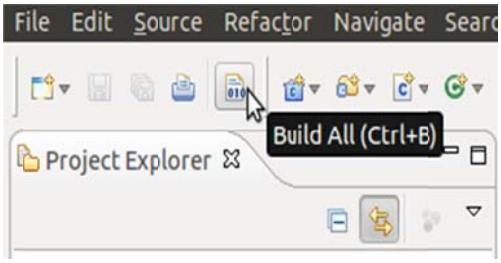


To Build All LinK+ Device Driver Projects click on **Build All** icon

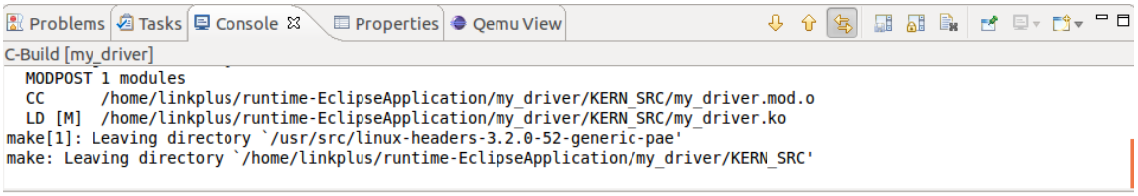




To build all C projects in workspace press **CTRL+B** or Click on **Build All** Toolbar.

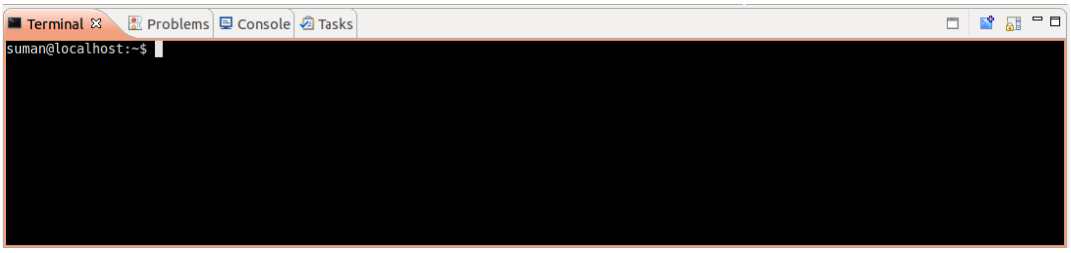


3.2.6.1 Build Output:



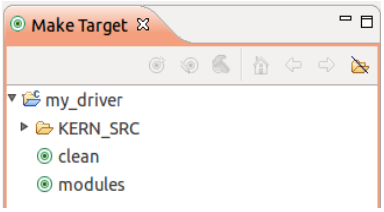
3.2.6.2 Eclispe Local Terminal Plugin :

To Open Linux Terminal in Eclipse IDE click on *Window -> Show View -> Other -> General -> Terminal*



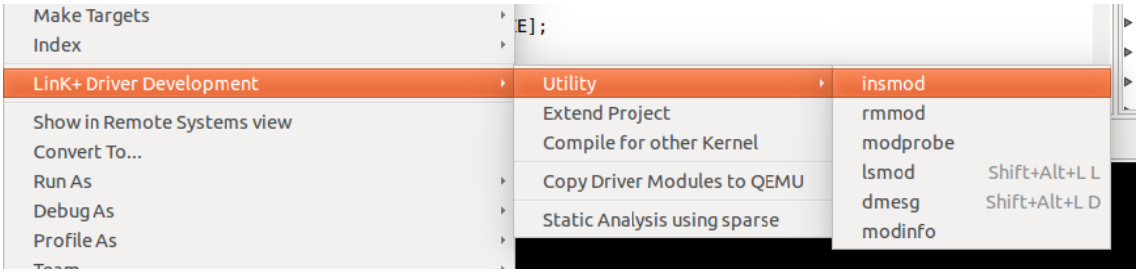
Note: For more information about Google ELT Project visit http://code.google.com/p/elt/

3.2.6.3 Make Target View:



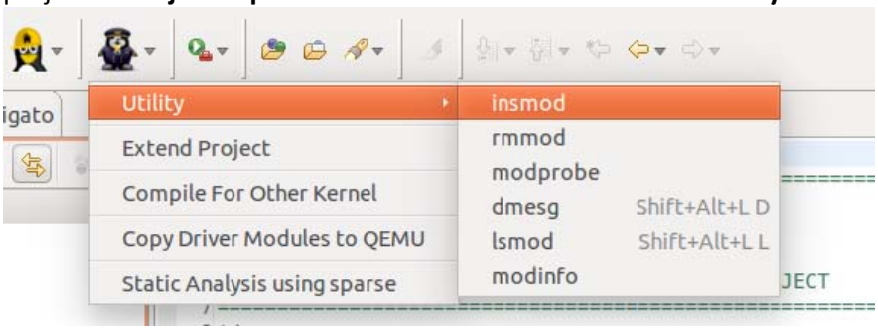
3.2.6.4 LinK+ Device Driver Command Execution:

Either Right Click on Project select **Link+ Driver Development**



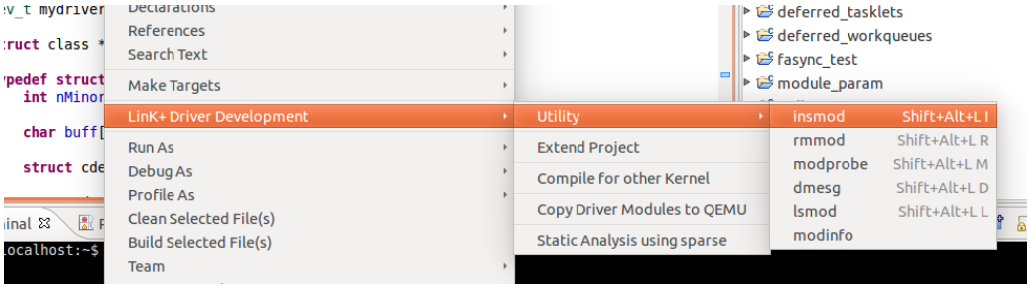
or

Select the project in **Project Explorer** View and click on LinK+ Driver **Utility** on Toolbar .



or

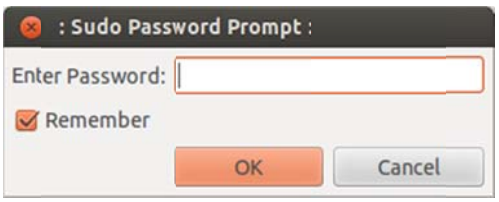
To run the Driver Build **Utility** for Driver Module Code directly from the Editor **Right Click** on the Editor



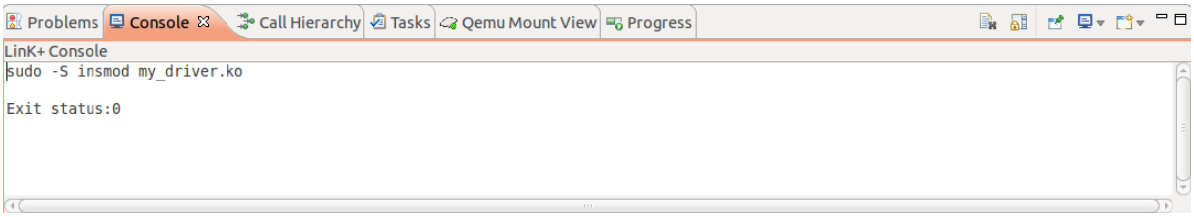
Note: Build Utility in Editor Context Menu is visible only when the C Editor contains a Linux driver module code.

Once user clicks on the utilities (insmod/rmmod/modprobe), the **Password Prompt** will be popped up.

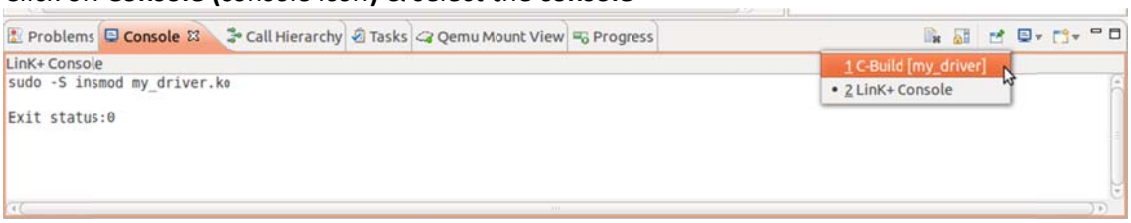
Sudo Password Prompt: To perform administrative tasks sudo password is required! (e.g, insmod, rmmod, modprobe)



3.2.6.5 Output Console:

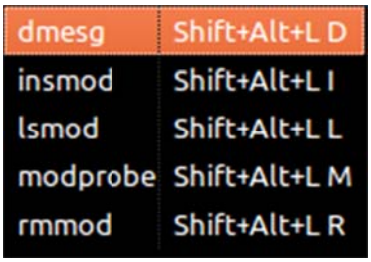


Switch between **LinK+ Console** and **C Build Console**
Click on **Console (***console icon***)** & select the **console**



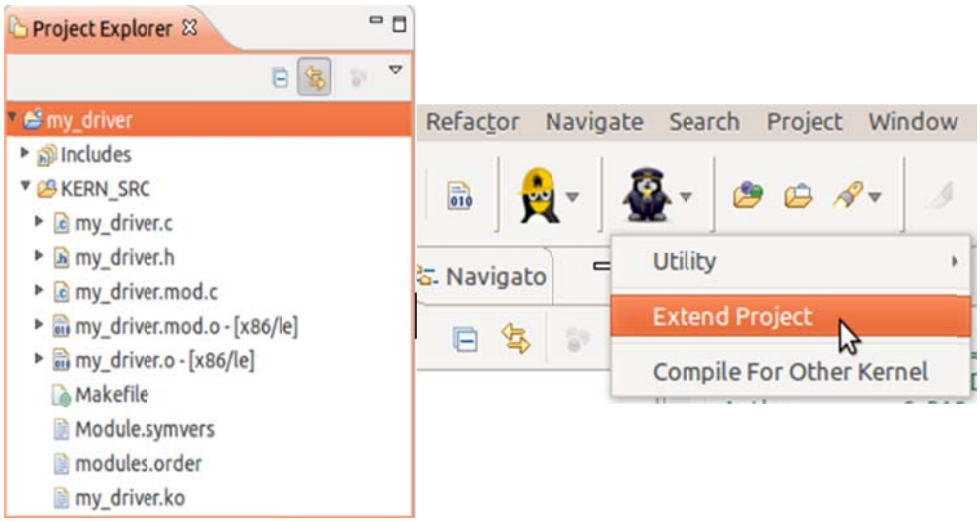
3.2.6.6 Short Keys

To get the device driver utilities (insmod, rmmod, modprobe, dmesg) with short keys, the user needs to press **SHIFT+ALT+L** and it will open the short keys window as shown in the screenshot below. Then the user needs to press
'd' for **dmesg**
'i' for **insmod**
'l' for **lsmod**
'm' for **modprobe**
'r' for **rmmod**

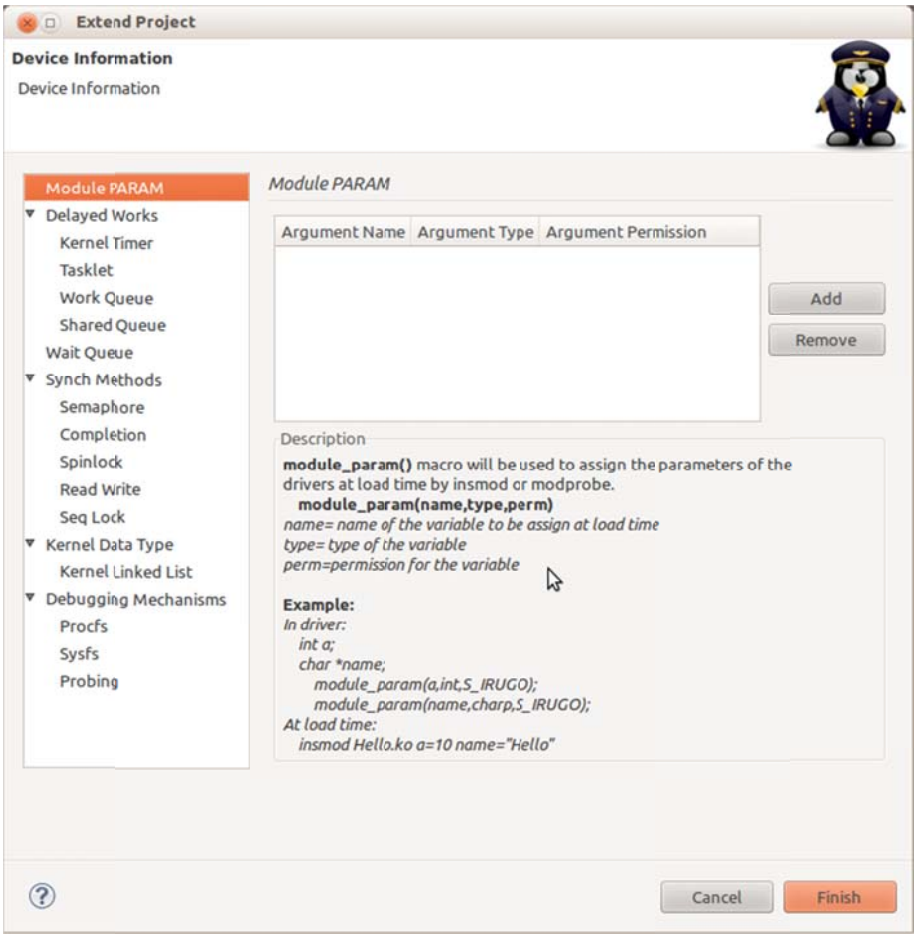


3.2.6.7 Extend Project:

Select the project in **Project Explorer View** and click on LinK+ Driver **Utility** on Toolbar .
Then, Click on **Extend Project**



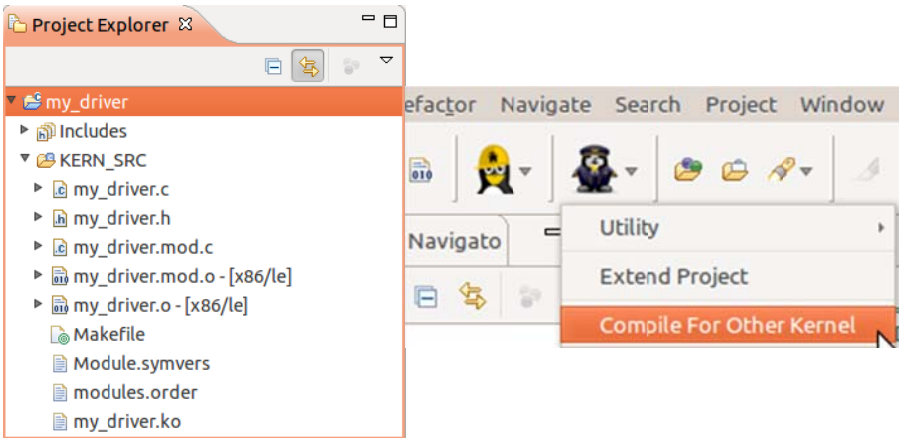
Select features you want to use. All features are optional. Click on **Finish** button



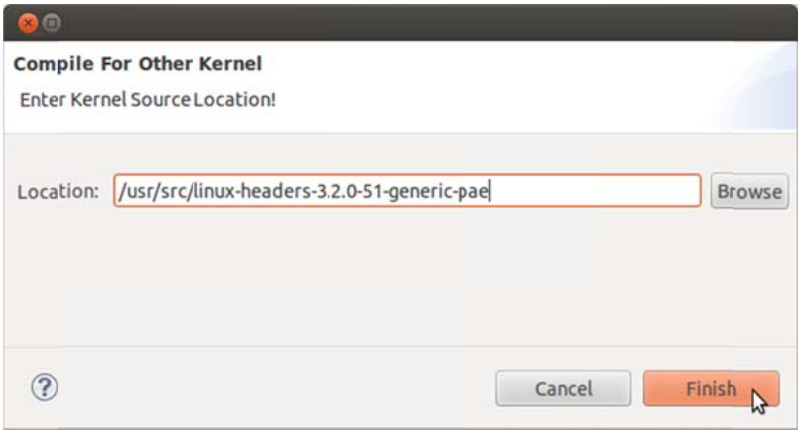
3.2.6.8 Compile For Other Kernel:

Select the project in **Project Explorer View** and click on LinK+ Driver **Utility** on Toolbar .

Click on **Compile For Other Kernel**



Click on **Browse** button and browse the Kernel Source location from file system.

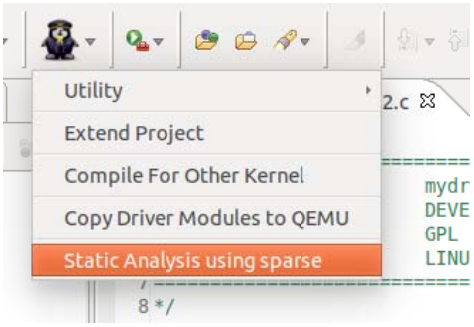


Click on **Finish** button

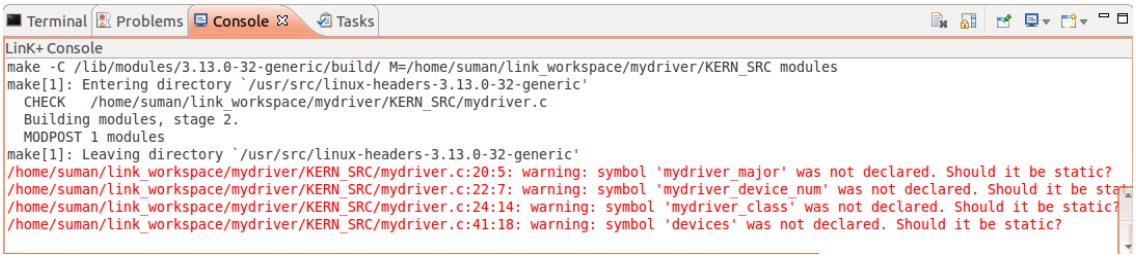
It will update the Makefile, project configuration and includes paths.

3.2.6.9 Static Analysis Using sparse :

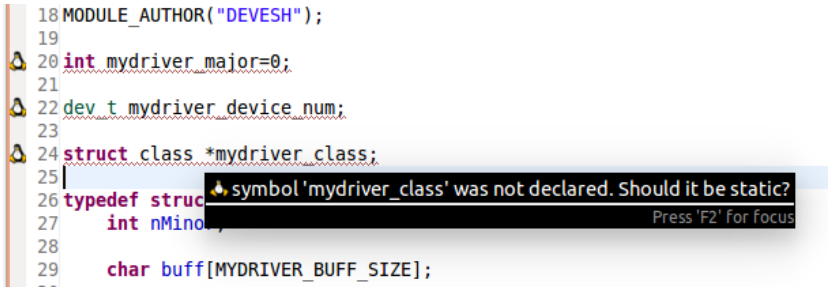
Click on LinK+ Driver **Utility** on Toolbar. Click on **'Static Analysis using sparse'**



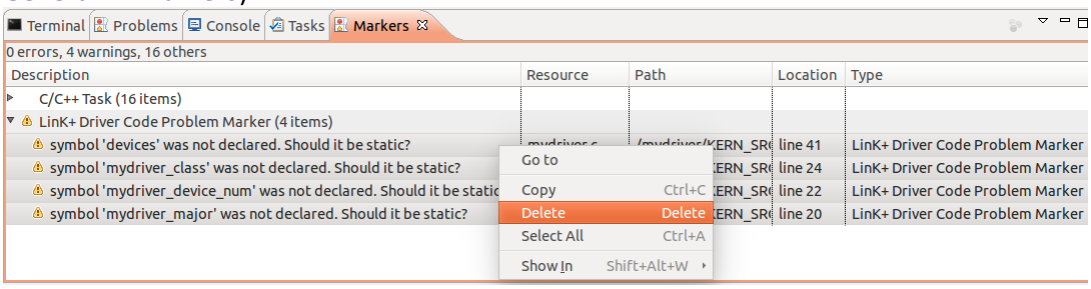
LinK+ Console shows sparse warnings.


Terminal
Problems
Console
Tasks
LinK+ Console
make -C /lib/modules/3.13.0-32-generic/build/ M=/home/suman/link_workspace/mydriver/KERN_SRC modules
make[1]: Entering directory `/usr/src/linux-headers-3.13.0-32-generic'
CHECK /home/suman/link_workspace/mydriver/KERN_SRC/mydriver.c
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory `/usr/src/linux-headers-3.13.0-32-generic'
/home/suman/link_workspace/mydriver/KERN_SRC/mydriver.c:20:5: warning: symbol 'mydriver_major' was not declared. Should it be static?
/home/suman/link_workspace/mydriver/KERN_SRC/mydriver.c:22:7: warning: symbol 'mydriver_device_num' was not declared. Should it be sta
/home/suman/link_workspace/mydriver/KERN_SRC/mydriver.c:24:14: warning: symbol 'mydriver_class' was not declared. Should it be static?
/home/suman/link_workspace/mydriver/KERN_SRC/mydriver.c:41:18: warning: symbol 'devices' was not declared. Should it be static?

and Marks all warnings in Editor



To delete LinK+ sparse Markers from Editor, Open Markers View (*Window -> Show View -> General -> Markers*)

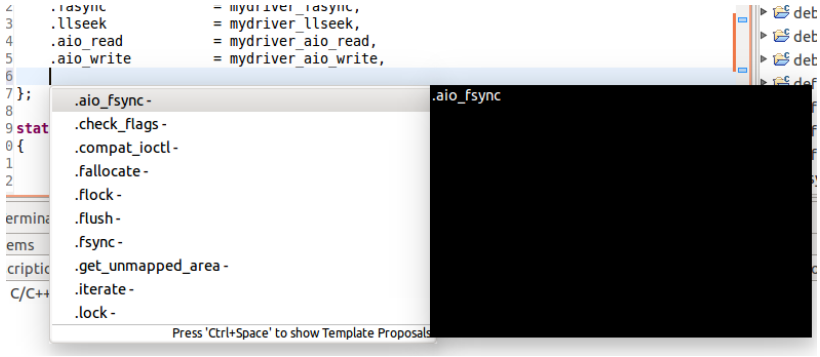


Right Click on the warning and click on **Delete**.

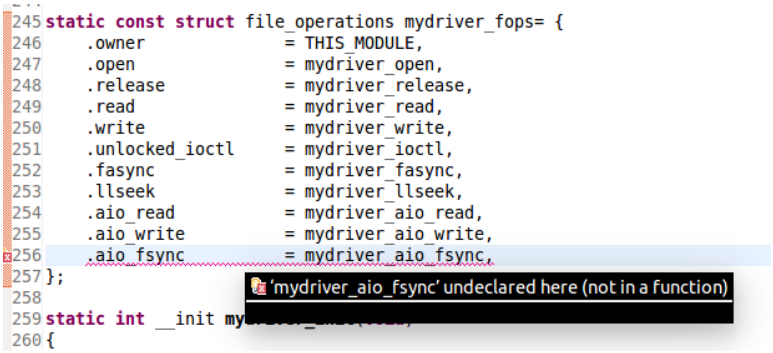
3.2.6.10 Content Assistance and Quick Fix Utility for LinK+ Driver Development

3.2.6.10.1 Quick suggestion for function pointers

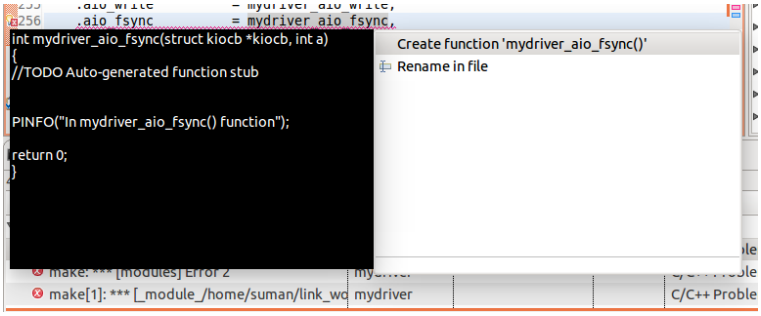
Press **'Ctrl+Space'** in function pointer structure to get suggestion for function pointers.



3.2.6.10.2 Function Pointer Stub Generation

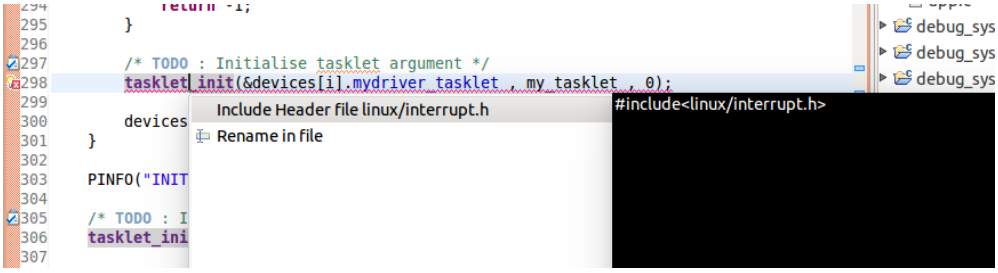


Press '**Ctrl+1**' or click on error marker icon



To apply suggestion in code, press **enter** key.

3.2.6.10.3 Header file name suggestion



To apply suggestion in code, press **enter** key.

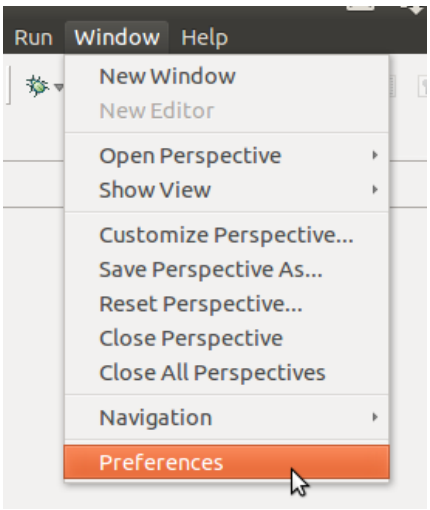
4

4 Linux Kernel Compilation Project Preferences

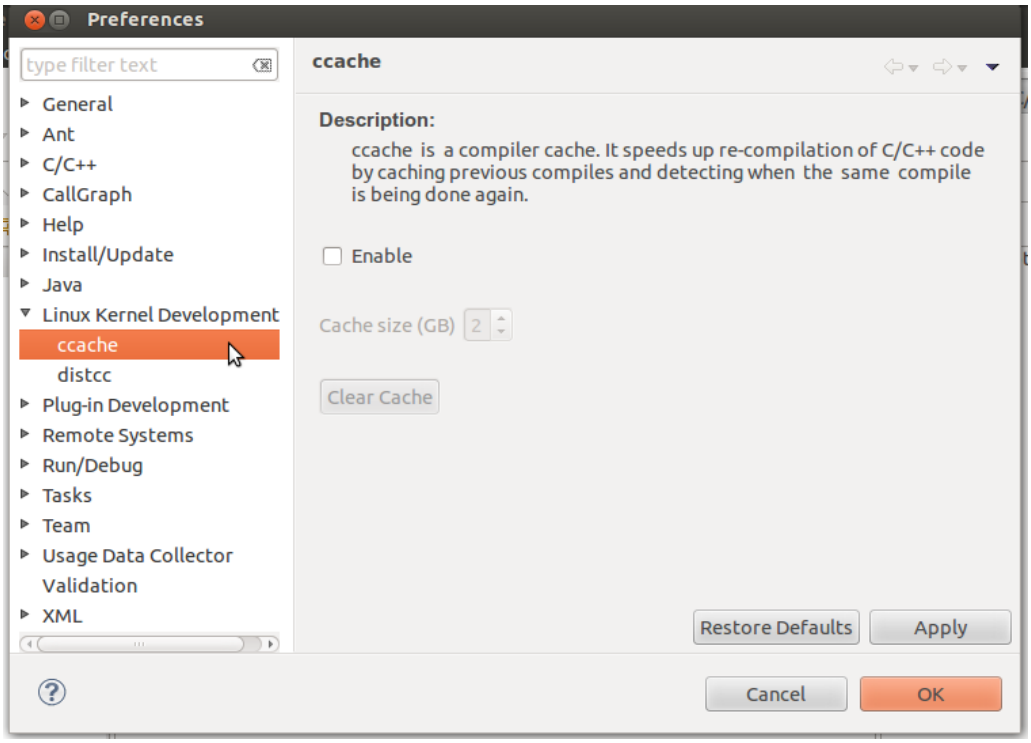
4.1 ccache Preference Settings

ccache speeds up the re-compilation of C/C++ code when the same compilation is repeated. This could come handy while recompiling the Linux Kernels again and again.

ccache can enabled by going through, *Window->Preferences*



In the Preference page, on the left pane, select *Linux Kernel Development->ccache*



In order to enable ccache, check the **Enable** option on the right pane. Once it is enabled, it checks whether ccache is installed on the machine or not. If not, a popup appears.

Click on **Yes** to follow up with the installation.

After the ccache installation is completed, you may change the ccache settings:

Cache size (GB): This option sets the maximum size for cache storage. The default and minimum size is 2GB and the maximum is 8GB.
Clear Cache: This button clears the entire cache, removing all cached files.
Restore Defaults: Reverts to the default settings.
Apply: Apply all the changes
OK: Apply the changes and exit the preferences page
Cancel: Do not apply the changes and exit.

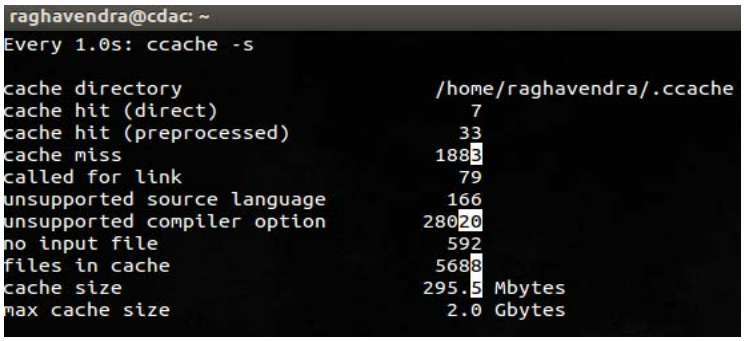
Now the user can compile the kernel image by Right clicking on the respective project in the **Project Explorer View** then goto

LinK to -> Compilation -> bzimage

You can monitor the progress of the ccache during the compilation on the terminal, using

watch -n1 -d ccache -s command

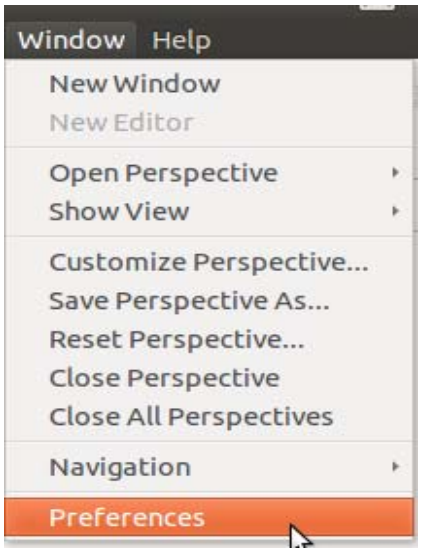
which opens a view as shown below, indicating the state of the ccache



4.2 distcc Preference Settings

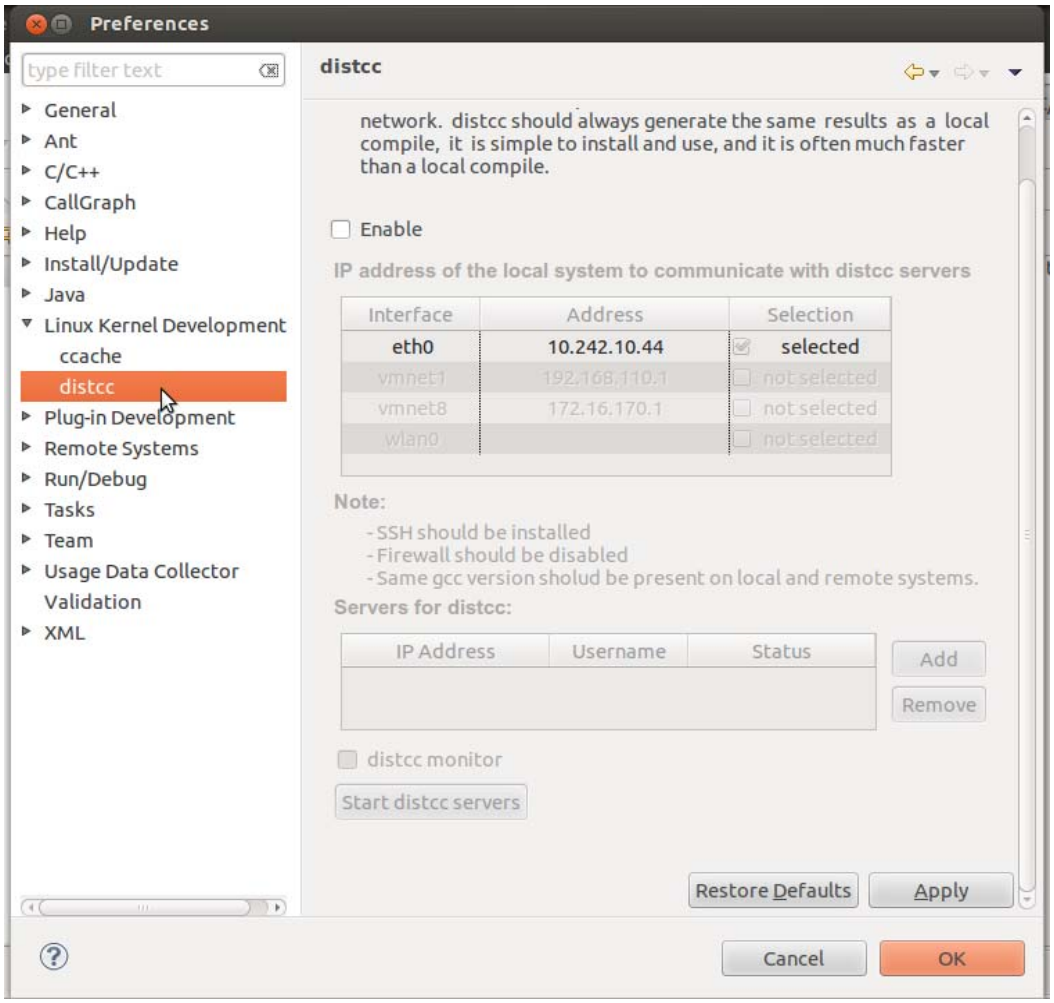
distcc is a program to distribute builds of C, C++, Objective C or Objective C++ code across several machines on a network. This could reduce the compilation time of huge build such as the Linux Kernel itself.

Distcc can be enabled through, *Window-> Preferences*



In the Preference page, on the left pane,

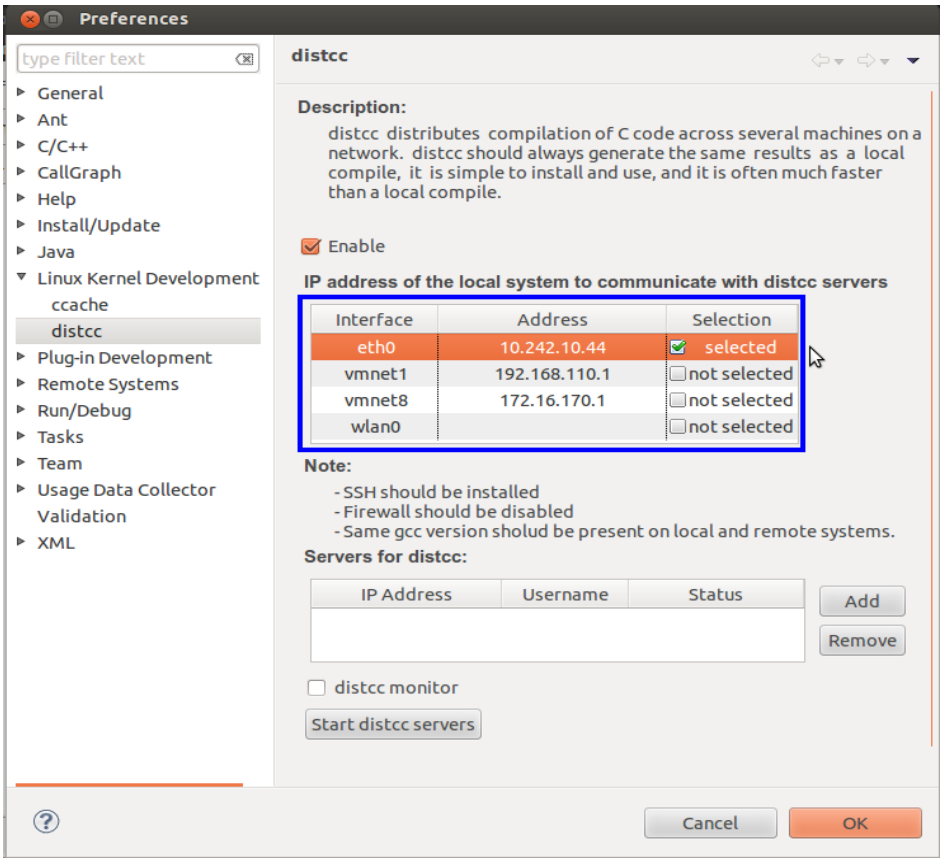
Select *Linux Kernel Development-> distcc*



In order to enable distcc, check the **Enable** option on the right pane. Once it is enabled, it checks whether distcc is installed on the machine or not. If not, a popup appears.

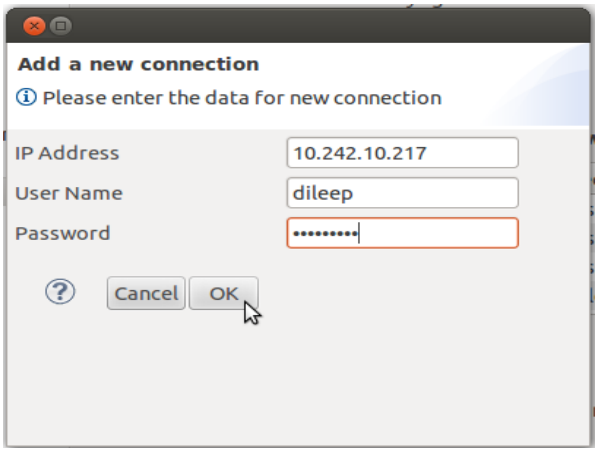
Click on **Yes** to follow up with the installation.

Once distcc is installed, you need to configure it. As highlighted below, select the interface of the local machine through which it should interact with the servers.



After you've decided the interface for interaction, you need to add the servers that help in the compilation process.

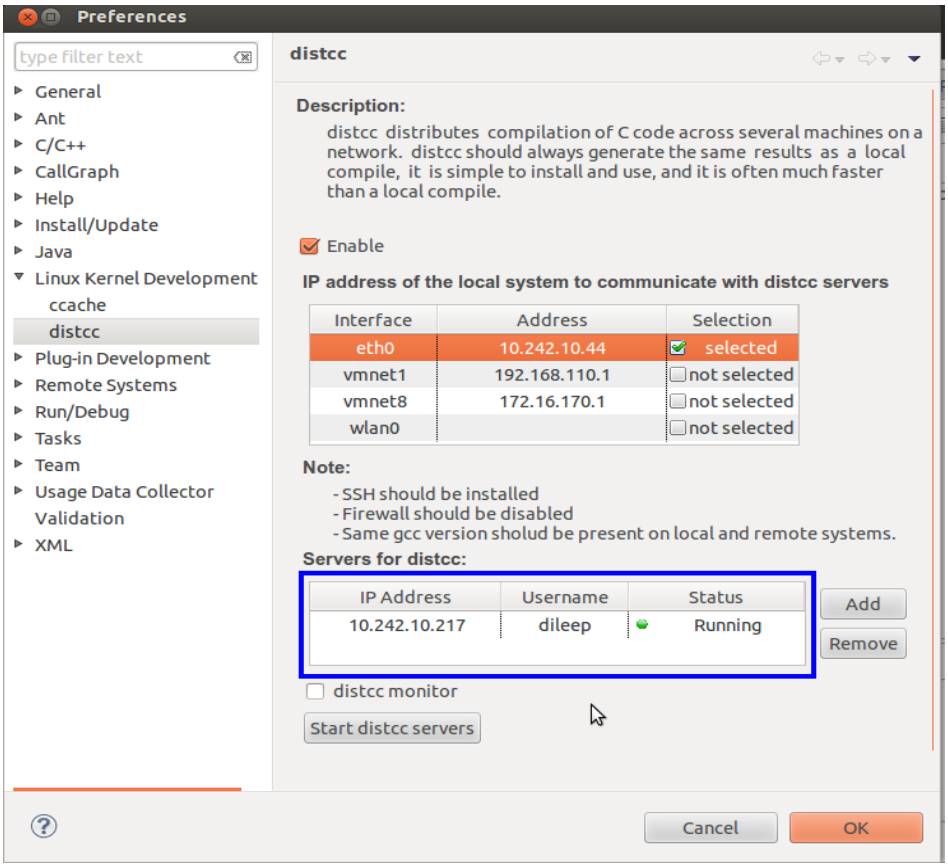
To do so, click the **Add** button and the popup appears as shown below:



IP Address: IP address of the machine (server) which helps in the compilation. **Username:** The username of the server (note that the user must have root privileges). **Password:** The password of the corresponding user

After filling up the details, click **OK**, which would configure the server (over ssh) and adds it

to the list of servers as indicated in the second table, as shown below



You can monitor the status of the server through the **Status** column. In the screenshot shown here, the status is shown as **Running**, indicating that the server is ready to accept connections.

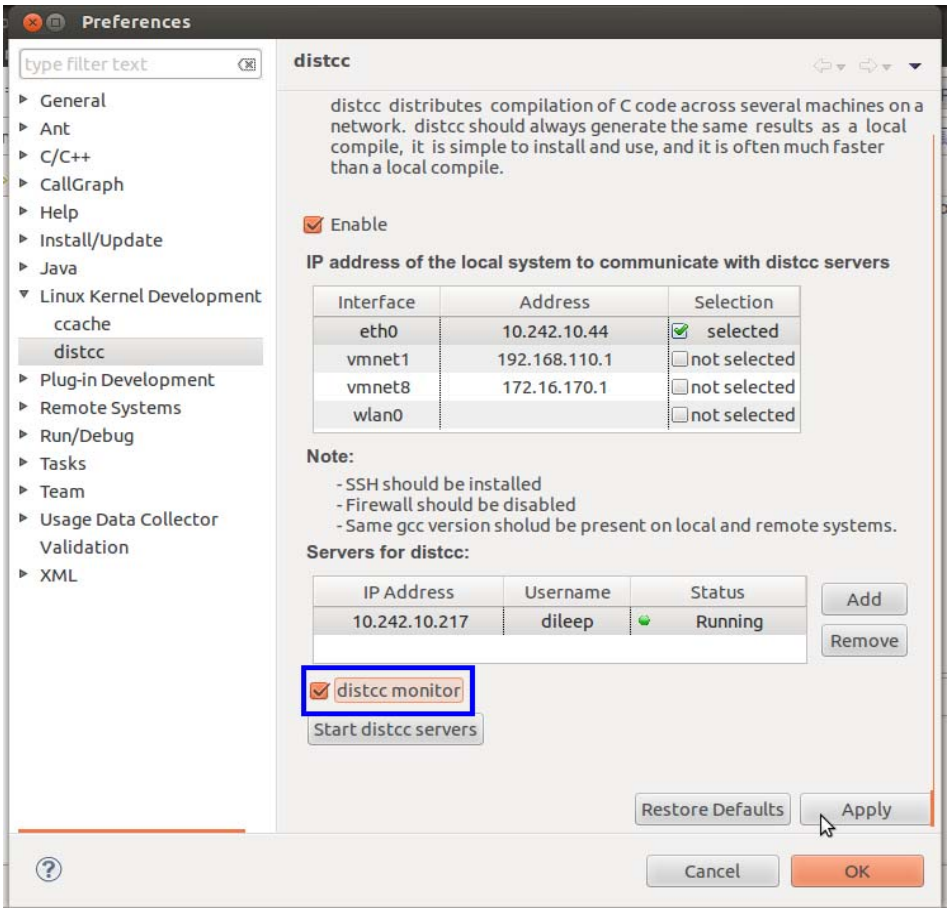
In case the status other than **Running**, you can click the **Start distcc servers** button to wake up the servers.

In any case if you wish to remove any server from the list, select the corresponding server and click the **Remove** button.

Now the user can compile the kernel image by Right clicking on the respective project in the **Project Explorer View** then goto

LinK to -> Compilation ->bzImage

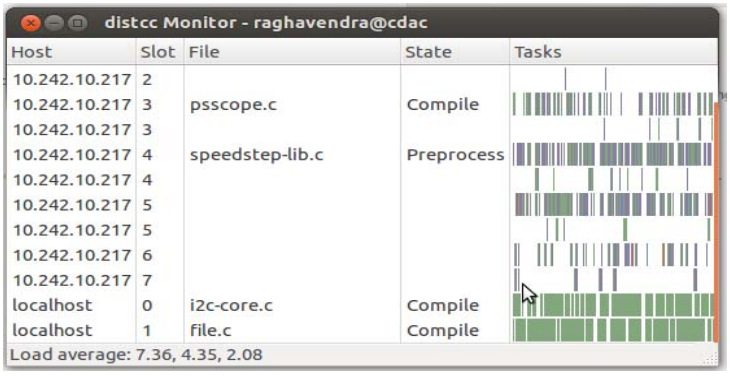
You can monitor the distributed compilation by checking the **distcc monitor** checkbox as shown below. The monitor is an application, which pops up while the compilation has started. Once you've enabled the monitor, it checks if the application is already present or not. If not, a prompt for installation will be popped and follow the steps to complete the installation.



Scroll down the page where you'll find the buttons **Restore Defaults** and **Apply**.

Restore Defaults: Reverts to the default settings i.e. clears the settings, even the servers list
Apply: Apply all the changes
OK: Apply the changes and exit the preferences page
Cancel: Do not apply the changes and exit.

Once the compilation starts, the distcc monitor appears (if enabled) as shown below



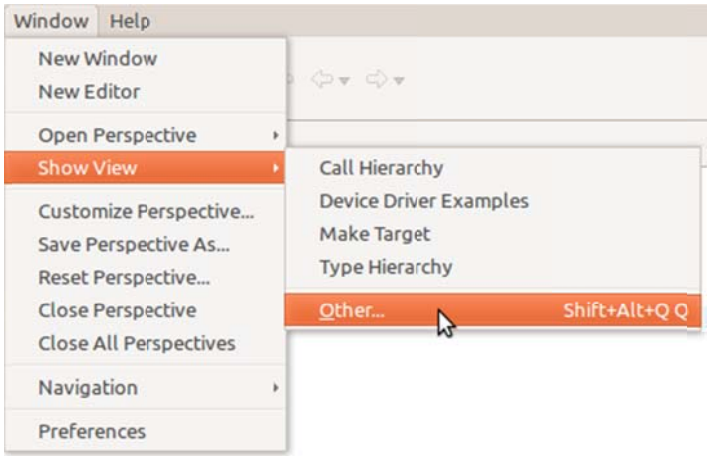
5

5 LinK+ Views

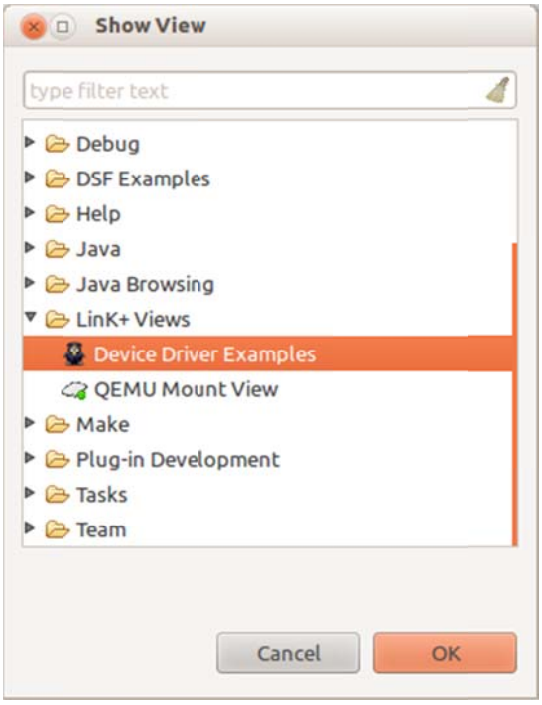
5.1 Linux Device Driver Example Programs

LinK+ IDE is also bundled with sample programs of Linux Device Drivers. To view the sample programs the user needs to go to the

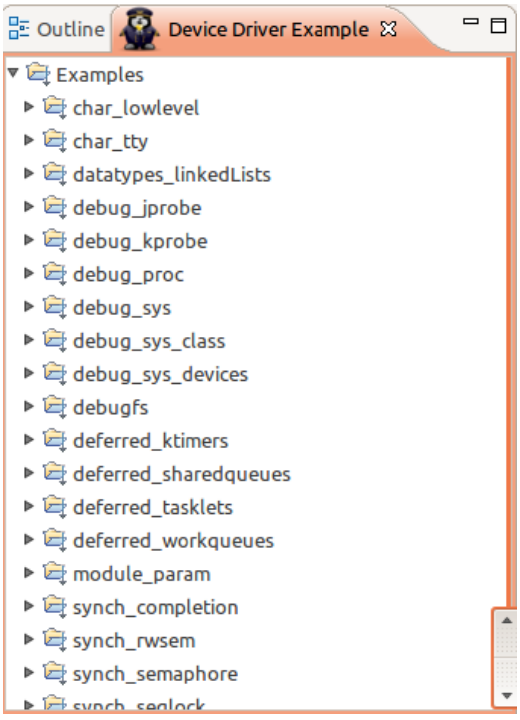
Window-> Show View->Other…



This will open a Show View wizard as shown below. Then user needs to go to the **LinK+ Views** & Click on the **Device Driver Examples**



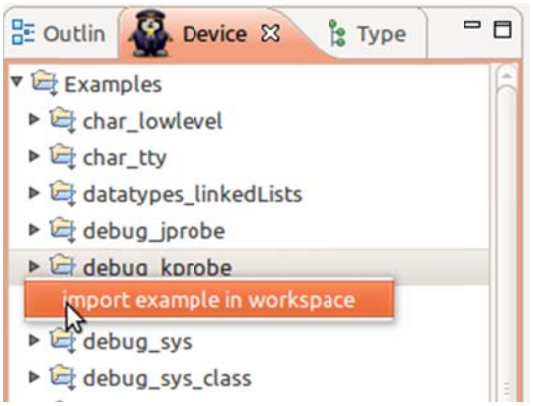
This will open **Device Driver Example View** as shown in the screenshot below. This View contains various types Device Driver examples. The user can select any example from the list given.



The user can also import the example program into the Workspace as a project by

Right Clicking on the **respective example program** & then, Click on the **Import example to workspace.**

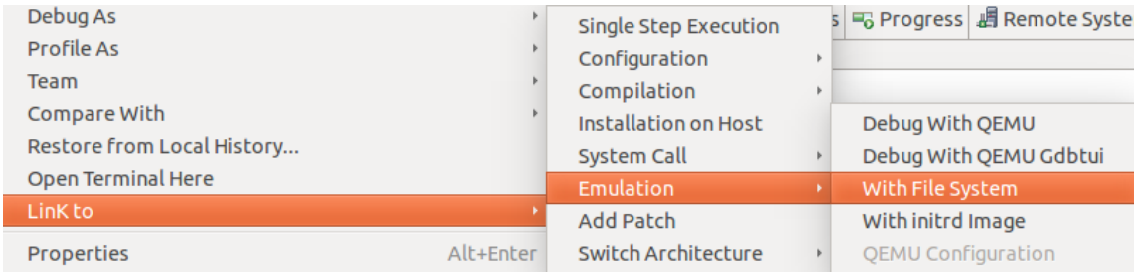
This example program will get imported into the Project Workspace where the user can extend the program based on his requirements



Note: *The example programs contain both the source code and the Readme document. In the Readme document the user can view the description and also the syntax & usage of that respective example*

5.2 QEMU Remote Systems

This section explores the Remote Systems view (Third party plug-in), which is generally used to monitor remote system. Here, we are making use of this to view and access QEMU's file system. The section describes the steps involved to setup the remote system view for QEMU and how to use it:

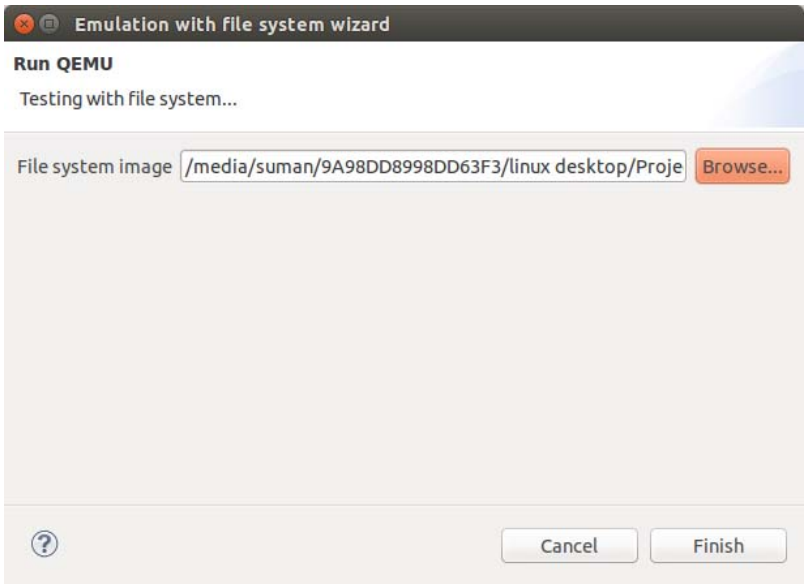


Select project in project explorer.

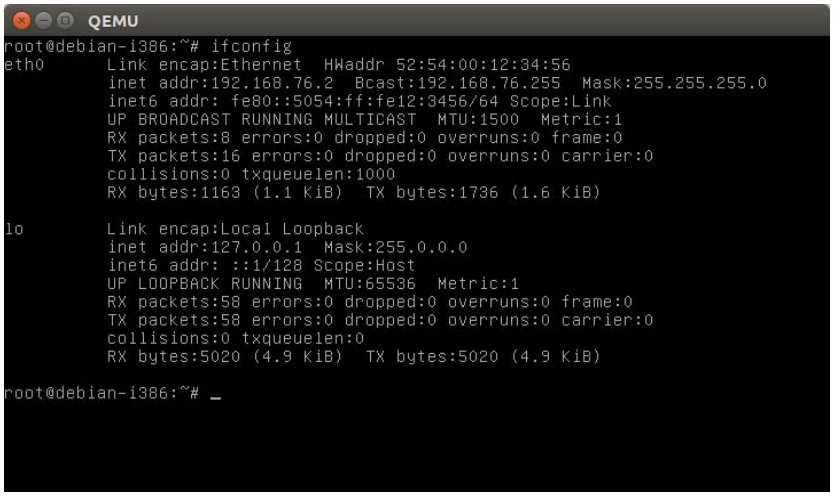
Navigate the following by right clicking on project

Goto **LinK to -> Emulation -> With File System**,

It will open the following window, browse for qemu image file and click Finish

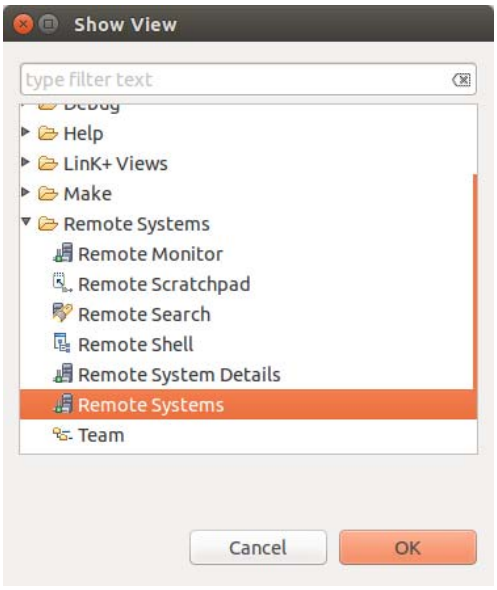


Linux will boot in QEMU file system. After booting note down the network address ip.

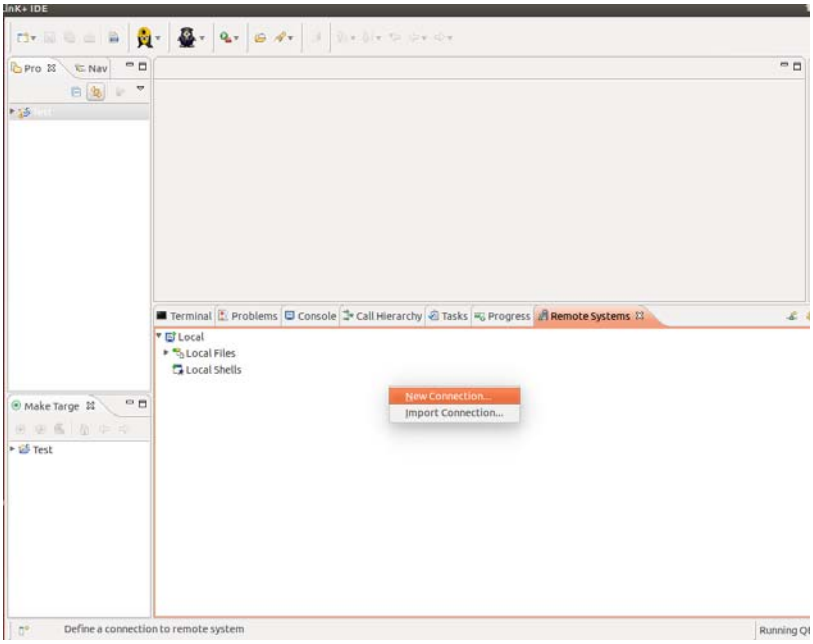

QEMU
root@debian-i386:~# ifconfig
eth0 Link encap:Ethernet HWaddr 52:54:00:12:34:56
inet addr:192.168.76.2 Bcast:192.168.76.255 Mask:255.255.255.0
inet6 addr: fe80::5054:ff:fe12:3456/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:8 errors:0 dropped:0 overruns:0 frame:0
TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1163 (1.1 KiB) TX bytes:1736 (1.6 KiB)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:58 errors:0 dropped:0 overruns:0 frame:0
TX packets:58 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:5020 (4.9 KiB) TX bytes:5020 (4.9 KiB)
root@debian-i386:~# _

In Eclipse goto

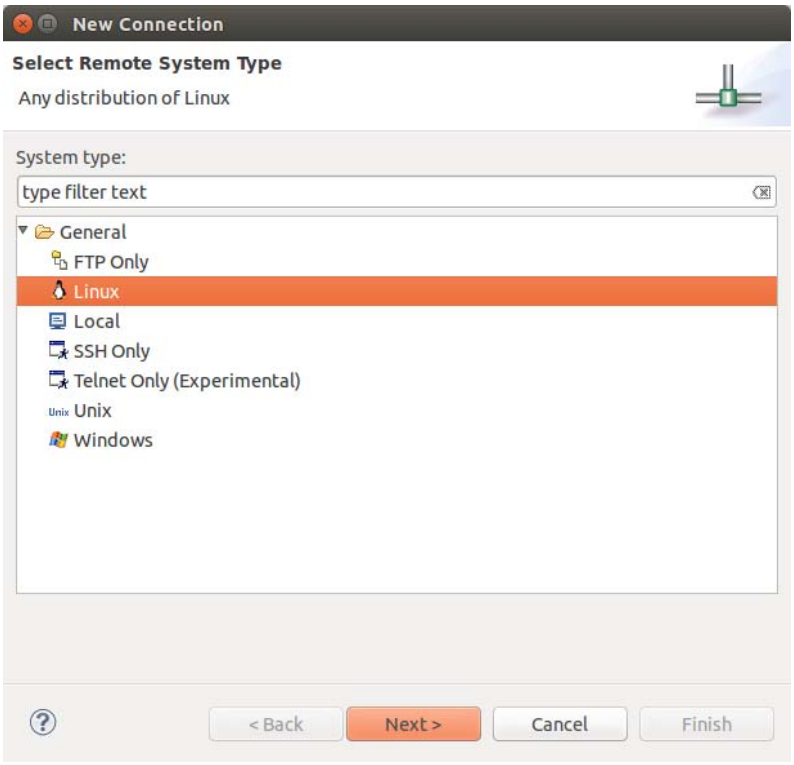
Window menu -> Show View -> Remote Systems menu -> Remote Systems



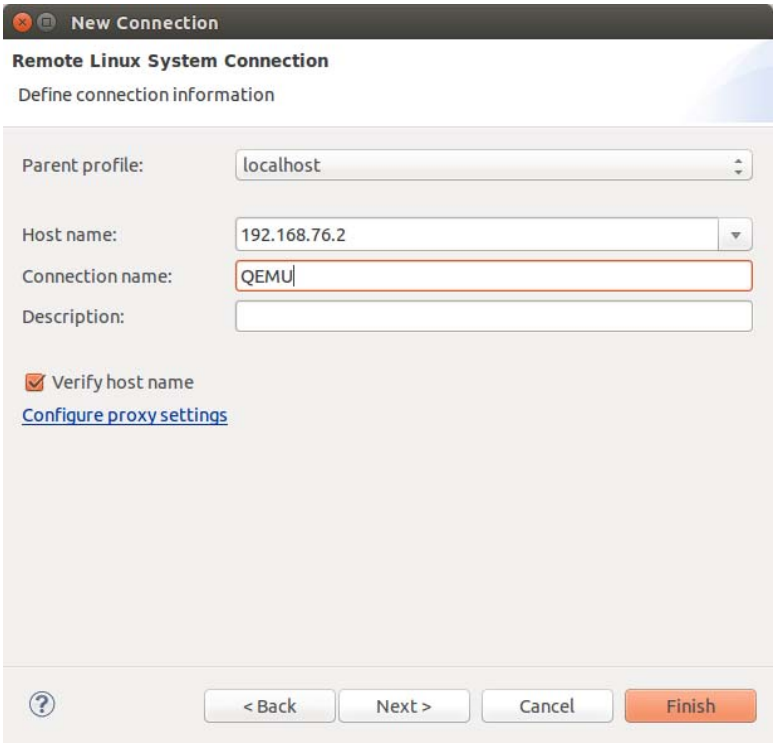
It will open Remote Systems View. Right click in the view and select New Connection... like the following.



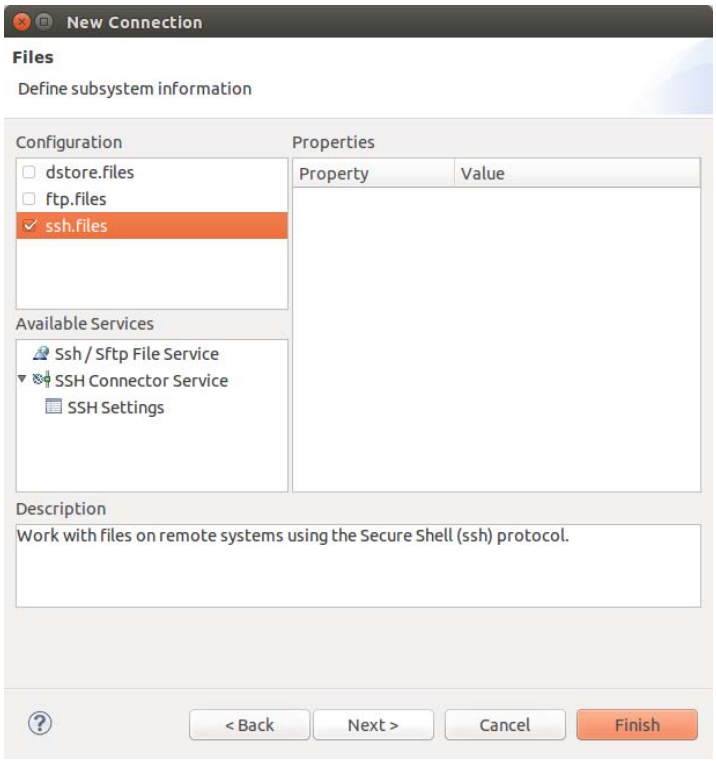
Select **Linux** and click **Next**



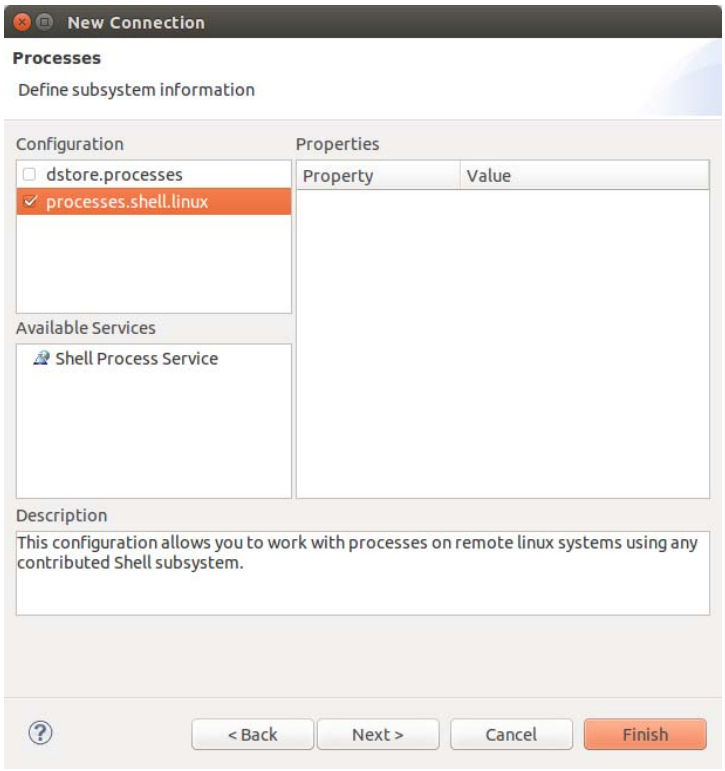
In Host name enter remote system IP address and enter any Connection name and click on Next



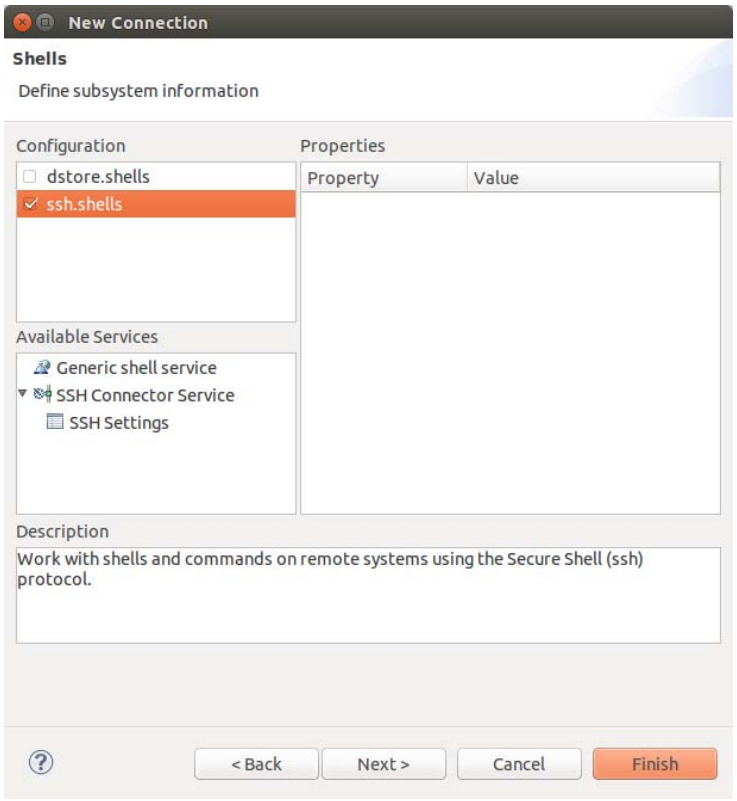
select **ssh.files** -> click **Next**



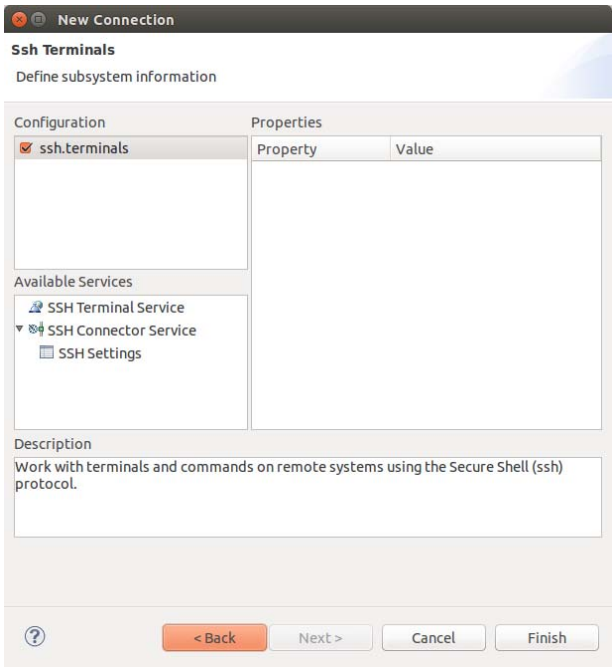
Select **processes.shell.linux** - > click **Next**



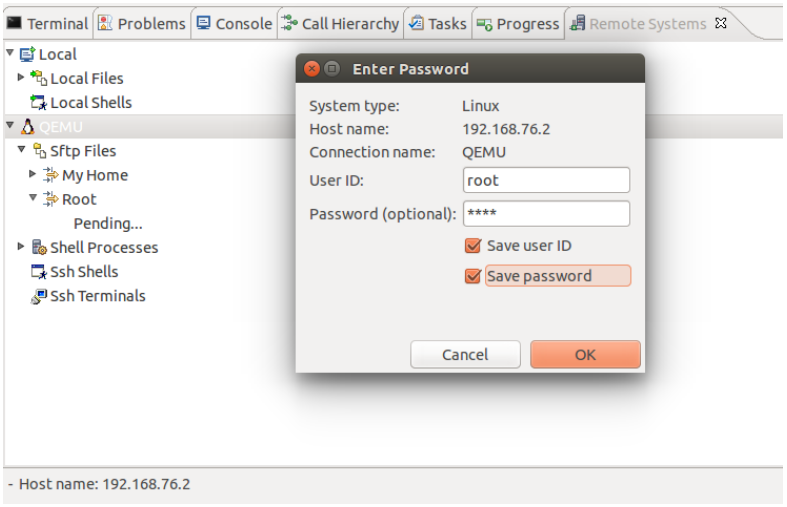
Select **ssh.shells** -> click **Next**



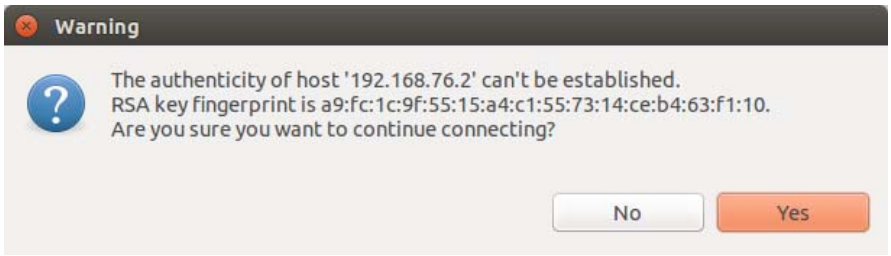
Select **ssh.terminals** -> click **Finish**



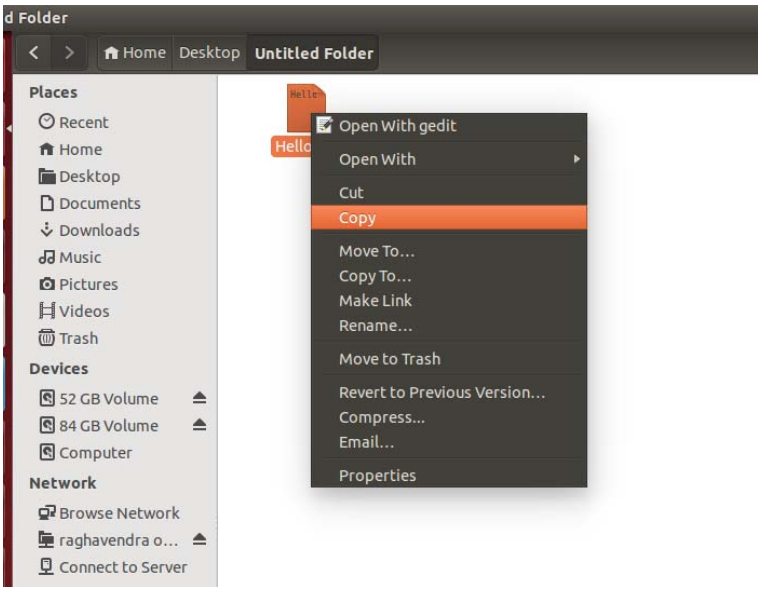
It will ask for the remote system credentials, enter required credentials and click on Ok. It will connect to the QEMU file systems.



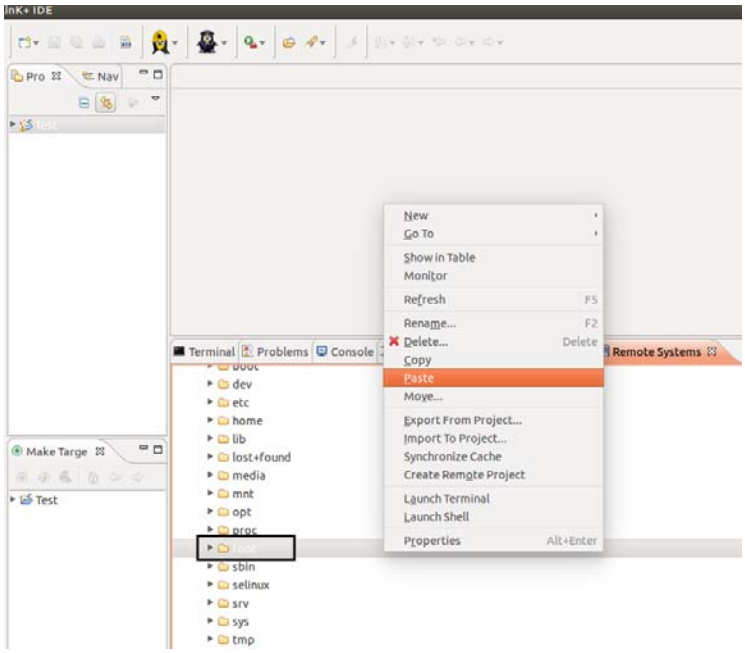
It will ask for authenticity for the first time click on **Yes** to connect.


Warning
The authenticity of host '192.168.76.2' can't be established.
RSA key fingerprint is a9:fc:1c:9f:55:15:a4:c1:55:73:14:ce:b4:63:f1:10.
Are you sure you want to continue connecting?
No
Yes

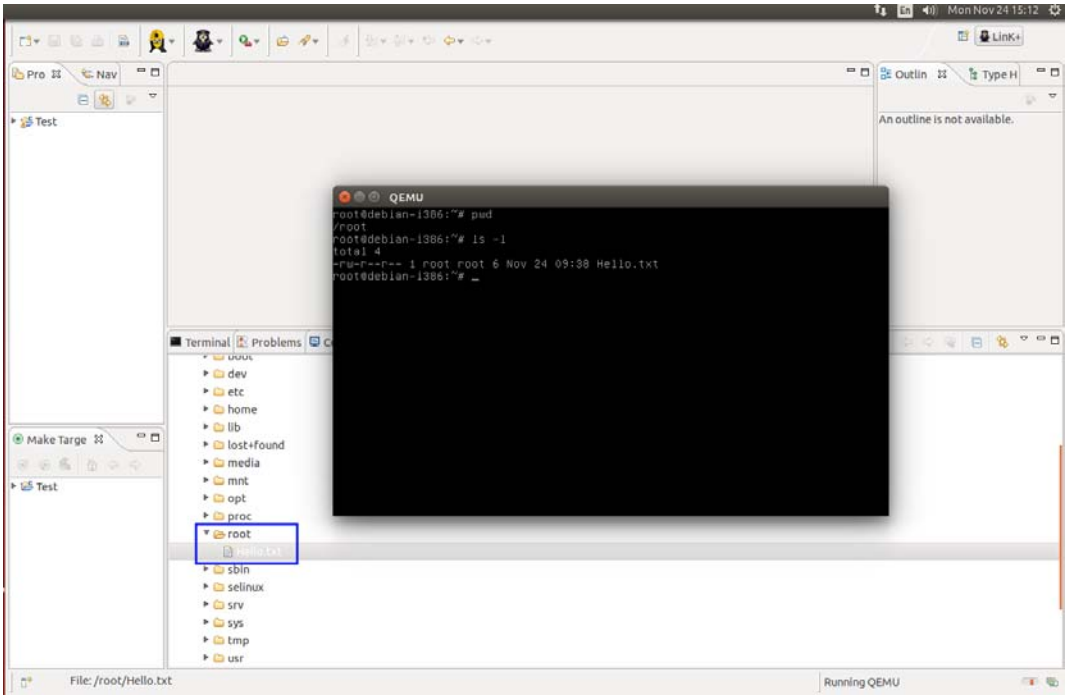
Select any file from file browser and copy that file



Select r**oot** folder in **Remote Systems** and Paste their



File will be copied into your Remote System and will show the file in Remote Systems file browser



6

6 Linux Kernel Emulation using QEMU

6.1 Linux Kernel Emulation using QEMU

After compiling Linux kernel image, QEMU emulator will be used to emulate the working of Linux kernel before installing on the host. For this process we have provided two options for emulation.

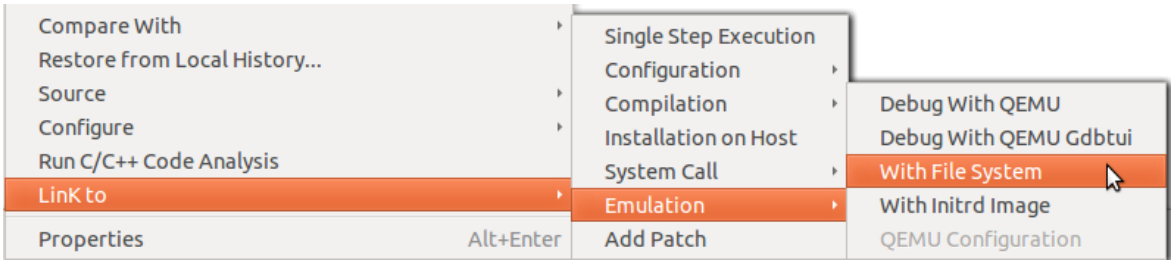
1. Linux Kernel Emulation using QEMU with File System
2. Linux Kernel Emulation using QEMU with initrd image

Note: *User can use any one of the options according to his requirement*

6.1.1 Linux Kernel Emulation using QEMU with File System

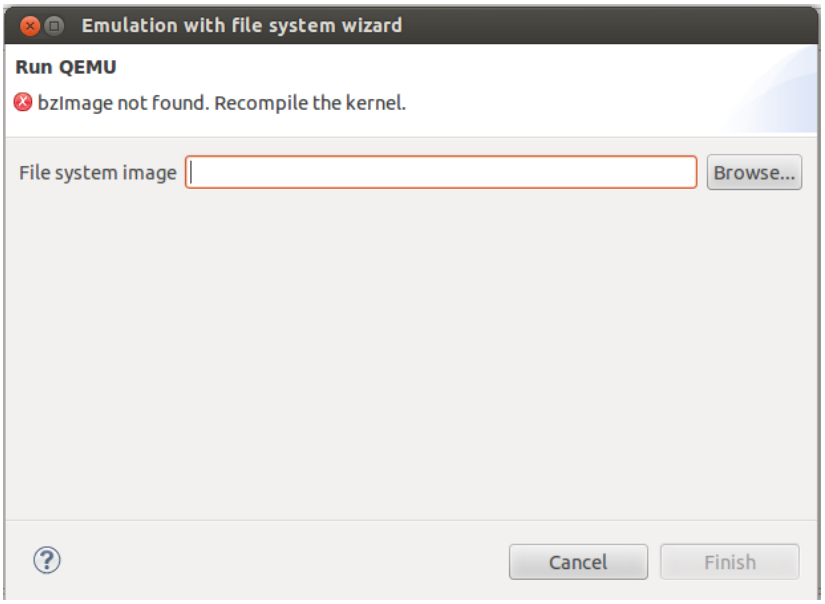
Right Click on the **Linux Kernel Project** in the **Project Explorer** and

goto *LinK to-> Emulation -> With File System*



It will open a new wizard as shown in the screenshot below.

Enter the **Password** and in the **File System image, Browse** for the Path of the qcow2 (qemu copy on write) File System image



> **Note:** *Along with the tool "qcow2" file system image will be provided*

After selecting the Path, Click on **Finish** button and it will start emulating Linux Kernel Image

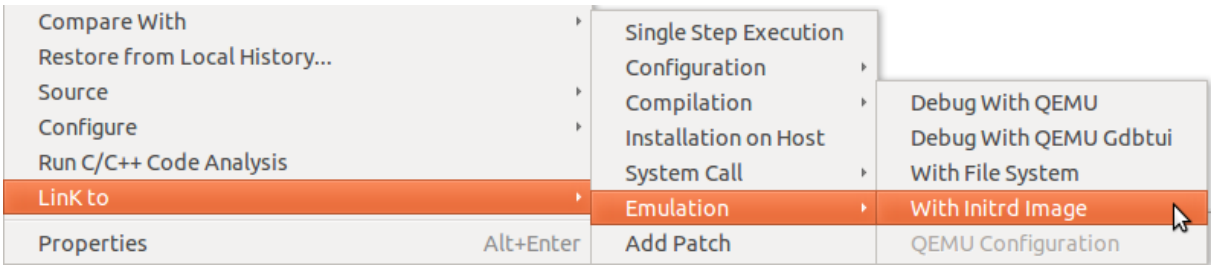
> **Note:** *After booting of Linux Kernel, the user needs to provide*
> ***Username:*** *root*
> ***Password:*** *root*

6.1.2 Linux Kernel Emulation using QEMU with initrd image

> **Note:** *To execute this option, initrd image should be generated for the Linux Kernel*

To Emulate Linux Kernel Image using initrd image, Right Click on the **Linux Kernel Project** in the **Project Explorer** and

goto *LinK to-> Emulation -> With initrd image*

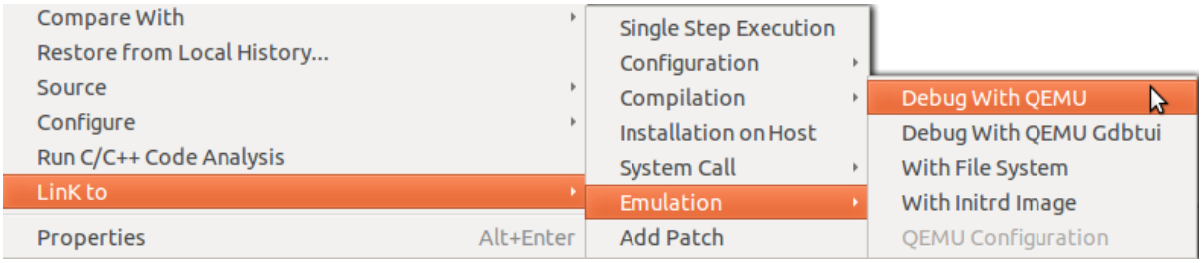


After selecting the path, Click on **Yes** button and it will start emulating Linux Kernel Image

6.2 Linux Kernel Debugging using QEMU

Debugging the Linux kernel image is possible with Eclipse CDT Debugger and QEMU Emulator. The user can avail the debugging option by

Right Click on the respective Project in the **Project Explorer** and goto *LinK to-> Emulation -> Debug with QEMU*



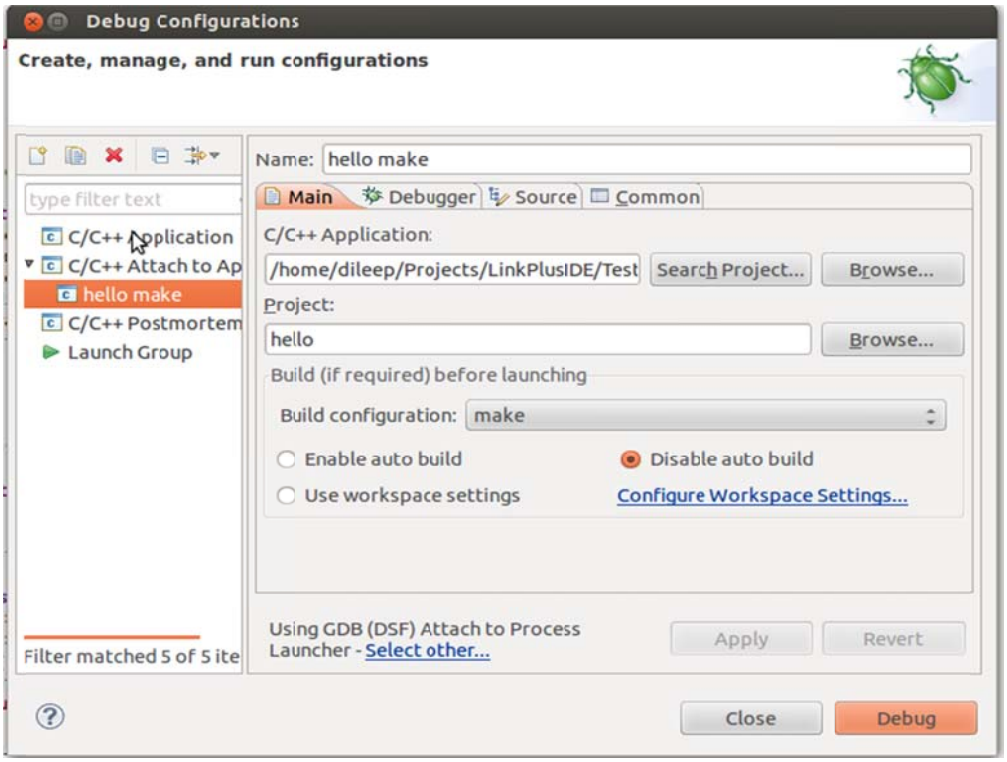
After click on the **Debug with QEMU** *and* follow the procedure,

First it will open the QEMU Emulator at **port number 1234**.

So After that it will ask for opening the Eclipse Debugger.

To open Eclipse Debugger Right Click on the respective Project in the **Project Explorer** and Click on *Debug As -> Debug Configuration*

It will open Debug Configurations Window as given below

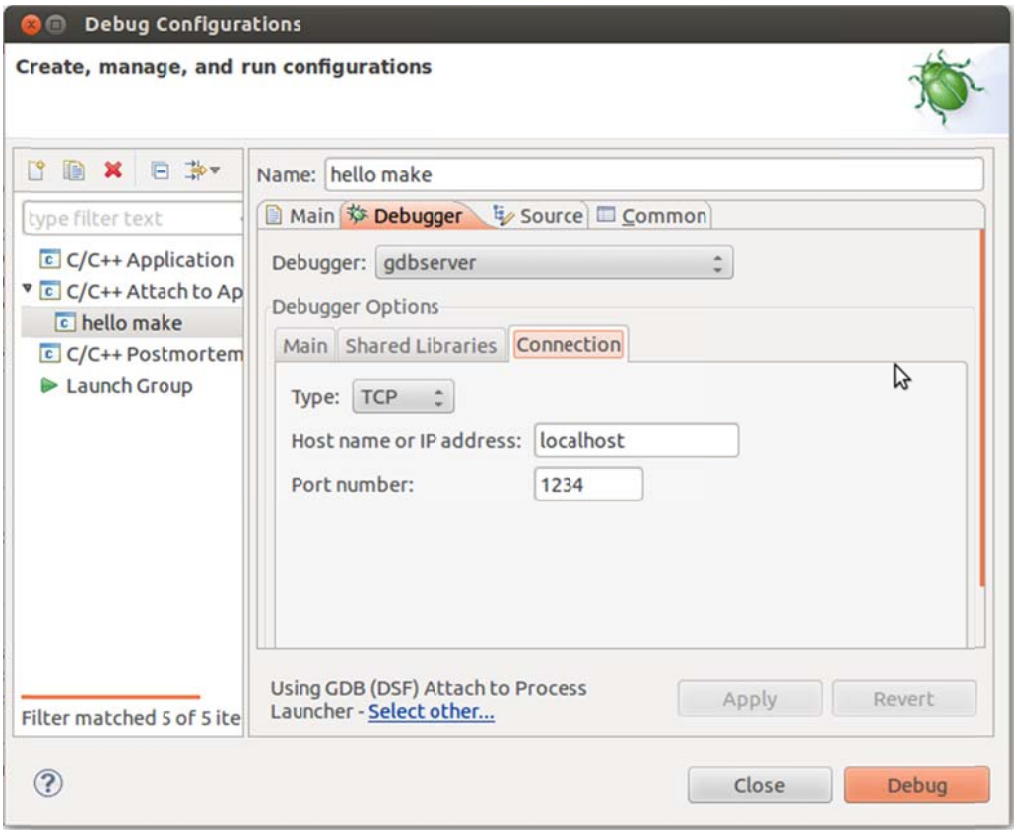


In the above window, on the left pane, Double Click on **C/C++ Attach to Application** it opens with Project name.

On the right pane, in the **Main tab**,

C/C++ Application: give the path of the “vmlinux” of the project and

Enable the **Disable auto build** option.
After the Main tab setting, in the same window, click on **Debugger Tab**



Debugger: select **gdbserver**
Debugger options: open the ***Connection* Tab** select

- **Type:** TCP
- ***Host Name or IP address:*** localhost
- **Port number:** 1234

Click on **Apply** button to apply changes, then click **Debug** button, it will ask for change to **Debug Perspective**. Click on **Yes** and start Debugging by giving breakpoint in **Console** Window

Eg:

b start_kernel
c

and then press **Enter** button

6.3 System Call Testing with QEMU

System Call testing on QEMU can be done by two steps.

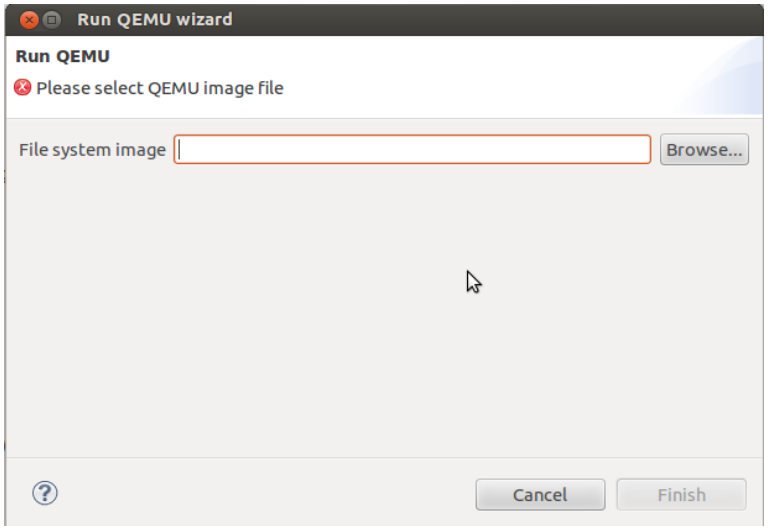
1. Run the newly added System Call kernel image with QEMU
2. Copy the System Call user application on to the running QEMU

The **first step** can be done by below given procedure:

Right Click on the Linux Kernel Project in the Project Explorer,

goto *LinK to -> System Call -> System Call testing with QEMU*

then it will open a wizard configuration



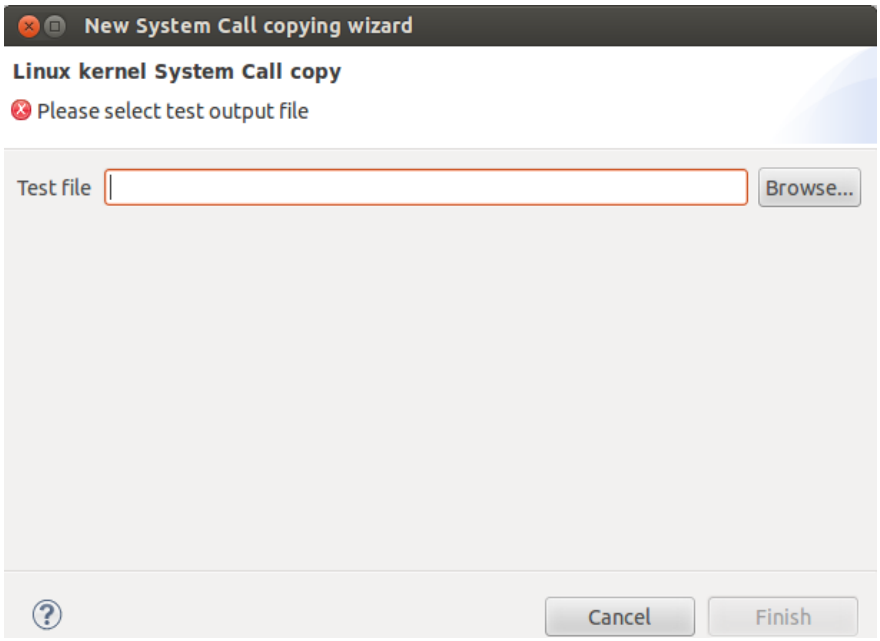
Provide **Password** and **Browse** for QEMU file system, click on **Finish** button.

The **second step** can be executed by the following given procedure:

After running QEMU, Right Click on the **Linux Kernel Project** in the **Project Explorer**,

LinK to -> System Call -> System Call app copy to QEMU.

It will open a wizard. **Browse** for target System Call application to copy onto the running QEMU and click on **Finish** button.



Note: *User can check the System Call application binary file in /tmp folder of the running QEMU file system*